Analyse und Modellierung einer Software zur Schnittstellenkommunikation für ein Raumfahrtprojekt unter Nutzung der TASTE Entwicklungswerkzeuge der ESA

BACHELORARBEIT

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Ulrike Witteck

14.09.2015

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	1952584, TINF12ITIN
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt, Berlin-Adlershof
Betreuer der Ausbildungsfirma	Dipl.-Ing.(FH) Gisbert Peter
Gutachter der Dualen Hochschule	Prof. Dr. Rainer Colgen

Eidesstattliche Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

_____

Berlin-Adlershof, 11. September 2015

Zusammenfassung

The ASSERT Set of Tools for Engineering (TASTE) ist eine Werkzeugsammlung zur Modellierung und Entwicklung heterogener, eingebetteter Echtzeitsysteme.

In der vorliegenden Bachelorarbeit wird die Nutzbarkeit der TASTE-Werkzeuge anhand eines Beispielprojektes des Instituts für optische Sensorsysteme des Deutschen Zentrums für Luft- und Raumfahrt getestet und bewertet.

Die standardisierte Schnittstellenkommunikation eines Teilsystems des Beispielprojektes wird mittels der Abstract Syntax Notation One (ASN.1) Notation modelliert. Anschließend wird die Nutzbarkeit der Modellierungssprache Specification and Description Language (SDL) zur Entwicklung von Raumfahrtprojekten anhand des modellierten Teilsystems analysiert. Das entstandene Modell wird mithilfe der TASTE-Entwicklungswerkzeuge simuliert und getestet. Zum Schluss wird eine Beschreibung zur Nutzung der TASTE-Werkzeuge beispielhaft für ein entsprechendes Softwareprojekt gegeben.

Ziel dieser Arbeit ist es herauszufinden inwieweit die TASTE-Entwicklungswerkzeuge für eine effiziente Softwareentwicklung, die die Qualitätsanforderungen der ESA erfüllt, geeignet ist. Zusätzlich soll darauf hingewiesen werden, worauf bei der Nutzung der TASTE-Werkzeuge zu achten ist.

Im Ergebnis der Bachelorarbeit wird deutlich, dass ein Teilsystem des Beispielprojektes, das die ESA-Standards zur Kommunikation erfüllt, mit den TASTE-Werkzeugen unter bestimmen Voraussetzungen modelliert und entwickelt werden kann. Der Einsatz der TASTE-Entwicklungswerkzeugen zeigt weiterhin, dass einige Punkte bei der Entwicklung einer Software mit den TASTE-Werkzeugen beachtet werden müssen und noch nicht alle Funktionen der Werkzeugsammlung einsatzfähig sind.

Alles in allem ist ein produktiver Einsatz der TASTE-Werkzeuge nur eingeschränkt möglich. Jedoch scheinen die TASTE-Werkzeuge ein zukünftig vielversprechendes Werkzeug zur Unterstützung der modellbasierten Softwareentwicklung zu werden.

Abstract

The ASSERT Set of Tools for Engineering (TASTE) is a toolset for the modeling and development of heterogeneous, embedded real-time systems.

In this bachelor-thesis the usability of the TASTE-toolset is tested and evaluated on the basis of an example project of the Institute of Optical Sensor Systems from the German Aerospace Center.

The standardized communication between the interfaces in the subsystem of the example is modelled with the Abstract Syntax Notation One (ASN.1) notation. Furthermore the usability of the Specification and Description Language (SDL) modeling language for the development of space systems is analyzed with the help of the modeled system. The model is then simulated and tested with the TASTE-tools. Finally a description is given for the use of the TASTE-toolset for the developing of corresponding software projects as an example.

The aim of this bachelor-thesis is to evaluate how suitable the development of software with the TASTE-toolset is and whether the software complies with the quality requirements of the ESA.

The result of the bachelor-thesis shows that it is possible to model and to develop a subsystem under certain conditions, which accomplishes the ESA requirements for communication, with the TASTE-toolset. In addition it shows that some points have to be considered while developing a software with the TASTE-toolset. Furthermore some functions of the TASTE-toolset are not or only usable with an extra effort from the developer point of view.

All in all one can say that a real usage of the TASTE-toolchain is restricted. However the TASTE-toolchain seems to become a promising tool to the support of the model based software development.

Inhaltsverzeichnis

Abbildungsverzeichnis **VII**

Tabellenverzeichnis **IX**

Quellcodeverzeichnis **X**

Abkürzungsverzeichnis **XI**

Vorwort **XIII**

1 Einleitung **1**

1.1 Motivation 1

1.2 Aufgabenstellung 3

1.3 Struktur der Bachelorarbeit 5

2 Modellgetriebene Softwareentwicklung **7**

3 On-Board Software des Projekts CHEOPS **11**

4 TASTE Entwicklungswerkzeuge **19**

4.1 Konzept der TASTE Entwicklungswerkzeuge 19

4.2 Ergänzende Werkzeuge 28

4.3 Alternative Modellierungswerkzeuge 29

5 Modellierung der Schnittstellenkommunikation **32**

5.1 Überblick der ASN.1 Notation 32

5.2 Modellierung einer typischen Protokollkommunikation 33

5.2.1 Erstellung des Datenmodells 34

5.2.2 Erzeugung des Interface Views 36

6 Modellierung der Softwarefunktionalität **42**

6.1 Überblick der Modellierungssprache SDL 42

6.2 Erstellung eines Beispiel-Softwaremodells mit SDL 44

7 Simulation und Test des Softwaremodells 56

7.1 Schedulability-Analyse mit Cheddar 57

7.2 Testen des erzeugten Programmcodes 59

8 Anwendung der TASTE Entwicklerwerkzeuge für das CHEOPS-Projekt 60

9 Fazit 64

9.1 Zusammenfassung 64

9.2 Ausblick 66

A Anhang XIV

A.1 Telekommandopaketstruktur XIV

A.2 Telemetriepaketstruktur XV

A.3 Ausschnitt aus der CHEOPS ASN.1-Spezifikation XVI

A.4 Interface View des Beispielprojektes XXI

A.5 „VerificationService“-Funktion XXIV

A.6 „CMDdispatcher“-Funktion XXVII

A.7 „HKservice“-Funktion XXVIII

A.8 „CheopsTempFPAccdControl“-Funktion XXXI

A.9 „FDIR“-Funktion XXXII

A.10 „DATAhandling“-Funktion XXXIV

Literaturverzeichnis XLIV

Abbildungsverzeichnis

1 Entwicklungsprozess mit dem MDA-Ansatz [5, S.7] 8
2 Characterising Exoplanet Satellite 11
3 CHEOPS SES-BEE Block Diagram [8, S.10] 12
4 CHEOPS APS-Komponenten [8, S.49] 13
5 CHEOPS Software Development Life Cycle [10, S.13] 18
6 Überblick der TASTE-Werkzeugsammlung [13, S.20] 20
7 TASTEGUI 21
8 Oberfläche des TASTE-Interface Views [11, S.3] 22
9 Oberfläche des TASTE-Deployment Views [11, S.4] 24
10 Marzhin Simulator [14, S.5] 25
11 Grafische Überwachung der Telemetriedaten in Echtzeit 27
12 Überwachung der Telekommando- und Telemetriepakete mit MSC Tracer . 27
13 Eigenschaften einer Systemfunktion 36
14 Automatisch generierte GUI zur Interaktion mit dem laufenden System . . 37
15 Editierung des Cyclic Interfaces 39
16 Eigenschaften der Schnittstellen 39
17 Beispiel eines SDL-Modells 43
18 Ausschnitt der GUI zum Senden der Telekommandopakete 44
19 SDL-Diagramm der „CMDhandling“-Funktion 45
20 SDL-Diagramm zur Verarbeitung eines „CMD_HK_Enable“-Paketes 48
21 SDL-Diagramm zur Verarbeitung der empfangenen Timer-Signale 49
22 SDL-Diagramm zur Verarbeitung eines „CMD_HK_Period“-Paketes 50
23 SDL-Diagramm zur Verarbeitung eines „CMD_HK_Disable“-Paketes 51
24 SDL-Diagramm der Prozedur „fget_temp_fpa_ccd_stub“ 52
25 Deployment View des Beispielsystems 56
26 Ausschnitt aus der Scheduling-Analyse mit Cheddar 58

27 Telekommandopaketstruktur nach dem CCSDS 203.0–B–2 Standard [9, S.42-44] XIV
28 Telemetriepaketstruktur nach dem CCSDS 102.0-B–5 Standard [9, S.46-48] XV
29 Interface View des Beispielsystems (Teil1) XXI
30 Interface View des Beispielsystems (Teil2) XXI
31 Interface View des Beispielsystems (Teil3) XXII
32 Interface View des Beispielsystems (Teil4) XXII
33 Interface View des Beispielsystems (Teil5) XXIII
34 SDL-Diagramm der „CMDdispatcher“-Funktion XXVII
35 SDL-Diagramm der „HKservice“-Funktion (Teil 1) XXVIII
36 SDL-Diagramm der „HKservice“-Funktion (Teil 2) XXIX
37 SDL-Diagramm der „HKservice“-Funktion (Teil 3) XXX
38 SDL-Diagramm der „CheopsTempFPAccdControl“-Funktion XXXI

Tabellenverzeichnis

1 CHEOPS APS-Tasks und deren Aufgaben [8, S.49-62] 14
2 Standardservices des ECSS–E–70–41A Standards [9, S.50] 15

Quellcodeverzeichnis

1 Auszug aus der CHEOPS DataView.asn XVI
2 VerificationService.c XXIV
3 FDIR.c XXXII
4 DATAhandling.c XXXIV

Abkürzungsverzeichnis

AADL	Architecture Analysis and Design Language
ACN	ASN.1 Control Notation
APS	Application Software
ASN.1	Abstract Syntax Notation One
ASN1SCC	ASN.1 Space Certifiable Copmiler
BEE	Back-End Electronics
CCSDS	Consultative Committee for Space Data Systems
CHEOPS	Characterising Exoplanet Satellite
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DM	Deadline Monotonic
ECSS	European Cooperation for Space Standardization
EEPROM	Electrically Erasable Programmable Read-Only Memory
ESA	European Space Agency
FEE	Front End Electronics
FPM	Focal Plane Module
GCC	GNU Compiler Collection
gcov	GNU Coverage
GUI	Graphical User Interface
ICD	Interface Control Document
IEC	Internationale Elektrotechnische Kommission
IOS	Informationsverarbeitung optischer Systeme
ISO	International Standards Organization
ISR	Interrupt Service Routines
ITU	International Telecommunication Union
MDA	Model Driven Architecture

MIAT	Minimum Inter-Arrival Time
MSC	Message Sequence Chart
NASA	National Aeronautics and Space Administration
OMG	Object Management Group
OS	Institut für optische Sensorsysteme
PBS	Primary Boot Software
PCU	Power Conditioning Unit
PI	Provided Interface
PIM	Platform Independent Model
PROM	Programmable Read-Only Memory
PSM	Platform Specific Modells
PUS	Packet Utilization Standard
RI	Required Interface
RMA	Rate Monotonic Analysis
RTDS	Real Time Developer Studio
RTOS	Real Time Operating System
RVS	Rapita Verification Service
SCU	Sensor Control Unit
SDL	Specification and Description Language
SEM	Sensor Electronics Module
SES	Sensor Electronics Sub-system
TASTE	The ASSERT Set of Tools for Engineering
UML	Unified Modeling Language
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
WCET	Worst Case Execution Time

Vorwort

Diese Bachelorarbeit ist im Deutschen Zentrum für Luft- und Raumfahrt (DLR) in der Abteilung Informationsverarbeitung optischer Systeme (IOS) des Instituts für optische Sensorsysteme (OS) im Rahmen des Studiengangs „Informationstechnik“ der Dualen Hochschule Baden-Württemberg Mannheim entstanden.

Das DLR ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Es besitzt nationale und internationale Kooperationen zur Forschung und Entwicklung in der Luft- und Raumfahrt sowie in Energie, Verkehr und Sicherheit.[1]

In OS werden neuartige abbildende optische Systeme entwickelt und im forschungsorientierten Umfeld, wie zum Beispiel auf Satelliten oder Flugzeuge eingesetzt. Diese Sensorsysteme müssen besonders hohe Anforderungen hinsichtlich Leistungsfähigkeit, Zuverlässigkeit und gewisser Qualitätsstandards, z. B. von der European Space Agency (ESA) oder National Aeronautics and Space Administration (NASA), erfüllen.[2]

Die Abteilung IOS hat sich auf die Entwicklung von Algorithmen und Software für Sensorsysteme für die optische Exploration spezialisiert. Die zu entwickelnden optischen Sensorsysteme werden auf der Grundlage systemtheoretischer Berechnungen und Modelle entworfen. Zur Bewertung bestimmter Systemparameter und Auswertealgorithmen, etc. der Systeme und zur Optimierung werden Simulationen durchgeführt.[3]

Die entwickelte Software muss den in der Raumfahrt üblichen Standards und Regeln des Softwareengineerings folgen, insbesondere dem ESA-Standard European Cooperation for Space Standardization (ECSS). Deren Einhaltung verursacht oft einen hohen Aufwand bei der Entwicklung und Verifikation solcher Softwareprodukte.[3] In dieser Arbeit wird eine Werkzeugsammlung betrachtet, die den Entwicklern die Entwicklung derartiger Systeme erleichtern soll.

An dieser Stelle möchte ich mich bei Herrn Ulmer für seine Hilfestellung und Unterstützung mit Fachwissen und Anregungen sowie bei Herrn Peter und Herrn Prof. Dr. Colgen für die Betreuung meiner Arbeit bedanken.

1 Einleitung

Die Motivation dieser Arbeit wird im folgenden Kapitel geschrieben. Diese enthält eine Einführung der Ausgangslage und der Problembeschreibung. Anschließend wird der Schwerpunkt der Arbeit und die Aufgabe benannt, die zur Lösung der Problematik beitragen soll. Nachfolgend wird ein Überblick zum Inhalt der einzelnen Kapitel dieser Arbeit gegeben.

1.1 Motivation

Im Institut für optische Sensorsysteme (OS) wurden bereits in den vergangenen Jahren u. a. optische Sensorsysteme für die Raumfahrt entwickelt. Diese Systeme enthalten immer mehr Software zur Datenverarbeitung und autonomen Steuerung der Systeme. Dabei ist es keine Seltenheit, dass diese Raumfahrtsysteme mehr als 15 Jahre im Orbit zuverlässig in Betrieb und wartbar sein müssen.

In den letzten Jahren ist die Komplexität von Softwareprojekten, wie sie auch in OS entwickelt werden, in der Funktionalität und im Design sehr gewachsen. Gleichzeitig sind die Qualitätsanforderungen zur Entwicklung, insbesondere dieser in Hardware eingebetteten Echtzeitsysteme ohne „sichtbare“ Schnittstelle zum Nutzer, gestiegen. Beispiele technischer Qualitätsanforderungen sind u. a. der Nachweis, dass keine Speicherlecks auftreten, das System eine definierte Antwortzeit hat (Schedulability-Analysis) oder der Softwarecode zu 100% getestet wurde (Code-Coverage). In der Regel ist eine solche Software für den sicheren Betrieb von Hardwaresystem im Wert von mehreren Millionen Euro verantwortlich und somit der Erfolg einer wissenschaftlichen Mission nicht zuletzt davon abhängig.

Die Erfüllung der hohen Qualitätsanforderungen ist sehr zeitaufwendig und ein Beweis zur Einhaltung der Anforderungen oft schwierig. Bei der herkömmlichen Softwareentwicklung für Raumfahrtprojekte werden strukturierte oder objektorientierte Ansätze verfolgt. Die Erstellung eines Softwaremodells erfolgt meist von der Softwareentwicklung praktisch

unabhängig und eine Codegenerierung aus dem Modell heraus ist nicht möglich. Der Nachweis zur Einhaltung der ESA-Standards erfolgt bei diesen Ansätzen durch umfangreiche Tests auf Unit- und Systemebene. Da bei der Erstellung der händisch ausgeführten Tests Fehler auftreten können, ist die Sicherstellung der Produktqualität nicht immer ausreichend gewährleistet.

Methoden des Softwareengineerings gewinnen deshalb zunehmend an Bedeutung. Vor allem modellbasierte Ansätze geraten immer wieder in den Fokus der Softwareentwickler. Das Ziel dieser Ansätze ist es, aus einem formalen Modell den Softwarecode zu generieren und somit die Portabilität, die Produktivität, etc. eines Systems zu steigern und den Entwicklungsaufwand zu reduzieren.

Aus diesem Grund versucht die ESA Methoden und Werkzeuge des modernen Softwareengineerings in der Entwicklung von Raumfahrtsystemen zu etablieren, die die Qualität und Effizienz der Softwareentwicklung erhöhen und den Entwicklungsaufwand verringern.

Dazu entwickelt die ESA zusammen mit weiteren Unternehmen die The ASSERT Set of Tools for Engineering (TASTE)-Entwicklungswerkzeuge. TASTE soll die Modellierung, Entwicklung und Verifikation von On-Board Software unterstützen und den Entwicklungsaufwand senken. Das Hauptkonzept von TASTE ist es, Systeme zu entwickeln, die durch ihren Aufbau richtig sind (engl. „correct by construction“), d. h., dass der Code das Modell nahezu identisch repräsentiert. Die Effizienz und Qualität der Softwareentwicklung mit TASTE wird durch standardisierte Verfahren und Schnittstellen zwischen den Werkzeugen erhöht. Zudem bieten die TASTE-Werkzeuge die Möglichkeit der Simulation und des Testens des entwickelten Systems, was den Entwicklungsaufwand bei fehlerfreier Entwicklungsumgebung stark reduziert.

Da in Projekten oft gleiche Standards beispielsweise zur technischen Schnittstellenkommunikation genutzt werden, ist es von Vorteil, wenn Softwaremodelle weitgehend projektunabhängig sind und somit wiederverwendet werden können.

Die Nutzung des ASN.1 Space Certifiable Copmiler (ASN1SCC), ein Werkzeug der

TASTE-Werkzeuge, wurde bereits erfolgreich anhand eines aktuellen Projektes der Abteilung IOS getestet. Im Rahmen dieser Arbeit soll der Einsatz weiterer TASTE-Werkzeuge zur Modellierung eines Raumfahrtsystems anhand eines Beispielmodells erprobt und bewertet werden.

1.2 Aufgabenstellung

Ziel der Arbeit ist es anhand des Beispielprojektes Characterising Exoplanet Satellite (CHEOPS), ein Teilsystem mit den TASTE-Werkzeugen zu modellieren. Dieses Beispielsystem soll eine Schnittstellenkommunikation nach den ESA-Standards realisieren, was einen Teil der gesamten Funktionalität darstellt. Dabei soll geprüft werden, wie und unter welchen Voraussetzungen die TASTE-Werkzeuge zur Entwicklung einer Software eingesetzt werden können, auch in Hinblick auf zukünftige Projekte.

Um eine Einschätzung der Nutzbarkeit der TASTE-Entwicklungswerkzeuge vornehmen zu können, soll zunächst eine typische Schnittstellenkommunikation eines Beispielprojektes mittels Abstract Syntax Notation One (ASN.1)[1] modelliert werden. Das Modell der Kommunikation soll auf dem Consultative Committee for Space Data Systems (CCSDS)/ Packet Utilization Standard (PUS)[2]-Standard basieren und weitgehend projektunabhängig sein, um eine Wiederverwendung zu ermöglichen. Über diese Protokolldefinition sollen die modellierten Systemfunktionen miteinander kommunizieren.

Anschließend soll die von den TASTE-Werkzeugen unterstützte Modellierungssprache Specification and Description Language (SDL) zur Entwicklung von On-Board Software für ein Raumfahrtprojekt analysiert werden. Dazu werden Systemfunktionen, die verschiedene Zustände besitzen können, in SDL implementiert. Es wird vorwiegend die grafische Notation der Modellierungssprache verwendet, da diese einen besseren Überblick über den Ablauf und das Verhalten einer Funktion gibt. Diese Systemfunktionen sollen bei-

[1] ASN.1 ist eine standardisierte Notation zur Darstellung von Datentypen.[4]

[2] CCSDS/PUS ist ein standardisiertes Kommunikationsprotokoll der ESA. Genauere Informationen sind in Kapitel 5.2 zu finden.

spielhaft verschiedene Services, die in einem weiteren ESA-Standard spezifiziert sind und im Beispielprojekt genutzt werden, implementieren.

Nach der Modellierung des Teilsystems folgen Simulation und Test mithilfe der TASTE-Entwicklungswerkzeuge. Diese sind beispielsweise Code-Coverage mit gcov, Performanceanalyse mit gprof, das Schedulability-Analysewerkzeug Cheddar. Um sicher zu gehen, dass die Simulation und der Test des Systems richtige Ergebnisse liefern, soll der entstandene Code des Softwareprojektes auf die reale Hardware portiert und dort getestet werden.

Schließlich sollen die Ergebnisse der Modellierungen mit den TASTE-Werkzeugen ausgewertet werden. Anhand dieser Auswertung soll ein möglicher Prozess zur Entwicklung eines Softwareprojektes beschrieben und auf folgende Fragestellungen eingegangen werden.

- Welche Schritte müssen für die Entwicklung einer Software mit TASTE ausgeführt werden?
- Welche Entwicklungsschritte können bereits mit den TASTE-Werkzeugen ohne Einschränkung umgesetzt werden?
- Welche Schritte können nur mit Abänderungen realisiert werden und wie sehen diese Änderungen aus?
- Welche Entwicklungsschritte können mit den TASTE-Werkzeugen nicht ausgeführt werden?
- Welche Entwicklungsfunktionen werden erst später in den TASTE-Werkzeugen integriert?

Dieser Entwicklungsprozess soll möglichst für zukünftige Projekte mit gleicher Problemstellung ebenfalls gelten.

1.3 Struktur der Bachelorarbeit

Neben der Einleitung im Kapitel 1 stehen im Kapitel 2 die Verwendung des Model Driven Architecture (MDA)-Ansatzes als eine mögliche Methode des Software Engineerings im Vordergrund.

Zum Test und zur Bewertung der Nutzbarkeit der TASTE-Entwicklungswerkzeuge wurde das CHEOPS Projekt gewählt, das ein aktuelles Projekt von OS ist. Kapitel 3 gibt einen Überblick über das CHEOPS Softwaredesign sowie über die von der ESA standardisierten Services, die das CHEOPS Projekt umsetzt. Dabei wird auf die zur Analyse der Modellierungssprache SDL beispielhaft implementierten Services genauer eingegangen.

Bevor die Entwicklung des Systems mit den TASTE-Werkzeugen beschrieben wird, werden diese in Kapitel 4 vorgestellt. Dieses Kapitel beschreibt, neben allgemeinen Informationen zu TASTE, den TASTE-Prozess, der bei der Entwicklung einer Software mittels TASTE durchlaufen werden muss und nach dem sich die folgenden Entwicklungsschritte des Beispielsystems richten. Dabei werden die wichtigsten Werkzeuge zur Softwareentwicklung vorgestellt. Neben den internen TASTE-Werkzeugen, können Werkzeuge, die nicht zu der TASTE-Werkzeugsammlung gehören, zur Softwareentwicklung mit TASTE genutzt werden. Zwei solcher Werkzeuge werden in Kapitel 4.2 erläutert. Anschließend werden zwei alternative Werkzeuge zu TASTE vorgestellt.

Die Modellierung der Schnittstellenkommunikation des Beispielprojektes wird mit den TASTE-Werkzeugen durchgeführt und in Kapitel 5 beschrieben. Zur Definition des Kommunikationsprotokolls wird die ASN.1 Notation verwendet. Einen ersten Überblick zu dieser abstrakten Notation bietet Kapitel 5.1. Diese Notation wird zur Erstellung des Datenmodells des Beispielsystems benötigt, in dem alle Datentypen des Systems definiert werden. Diese Datentypen werden wiederum zur Modellierung der Systemfunktionen und Schnittstellen genutzt. Die Spezifizierung des Datenmodells und die Definition der Systemfunktionen und Schnittstellen für das Beispielsystem wird in Kapitel 5.2 thematisiert.

Die Systemfunktionen werden nach ihrer Definition in SDL und C implementiert.

Kapitel 6.1 gibt einen Überblick über die Modellierungssprache SDL. Die Systemfunktionen sollen beispielhaft einige der Services modellieren, die von dem Beispielprojekt CHEOPS genutzt werden. Die Funktionalität der Systemfunktionen wird in Kapitel 6.2 erläutert und die Nutzbarkeit der Modellierungssprache SDL anhand einiger dieser Funktionen analysiert.

Nach der Modellierung des Systems folgen Simulation und Test. TASTE bietet die Möglichkeit eine Schedulability-Analyse des Systems mittels Cheddar durchzuführen. Mit einer Schedulability-Analyse können Aussagen über die zeitliche Ausführbarkeit und die vordefinierte Antwortzeit des Softwaresystems getroffen werden. Das Analysewerkzeug Cheddar und das Resultat der Schedulability-Analyse des Beispielsystems mit Cheddar wird in Kapitel 7.1 beschrieben. Anschließend werden die Testmöglichkeiten der TASTE-Werkzeuge in Kapitel 7.2 benannt.

Es folgt die Beschreibung eines möglichen Prozesses zur Entwicklung eines konkreten Softwareprojektes. Kapitel 8 beschreibt diesen anhand des CHEOPS Projektes und verdeutlicht, welche Entwicklungsschritte bereits möglich sind, was bei der Nutzung der TASTE-Entwicklungswerkzeuge beachtet werden muss und welche Funktionen noch nicht vollständig integriert sind.

Schließlich werden die Ergebnisse in Kapitel 9 zusammengefasst. Es wird dargestellt, inwieweit die TASTE-Werkzeuge zur Modellierung eines Raumfahrtsystems geeignet sind. Einen Ausblick für die weitere Verwendung der TASTE-Werkzeuge zur Entwicklung von Raumfahrtsoftware wird am Schluss dieser Arbeit gegeben.

2 Modellgetriebene Softwareentwicklung

Eine Herausforderung in der Softwareentwicklung sind immer größer und komplexer werdende Softwareanwendungen. Zusätzlich werden Qualitätsanforderungen an die Softwareentwicklung immer höher.

Manuelle Programmierung des Entwicklers wird dadurch stets aufwendiger und zeitintensiver. Zudem werden Fehler bei der manuellen Programmierung oft erst sehr spät entdeckt und die Korrektur wird demnach sehr aufwendig. Entwickler verlieren wertvolle Zeit bei der Suche nach gerade bei eingebetteten Systemen schwer zu findenden Fehlern. Der Entwicklungsprozesse wird infolgedessen verzögert und Fertigstellungstermine können nicht gehalten werden. Der Programmcode ist für Nicht-Entwickler kaum lesbar und nur der Programmierer selbst besitzt einen Überblick. Oftmals leidet die Qualität der Software darunter.

Ein weiteres Problem ist die Verwendung unterschiedlicher Werkzeuge in einem Team. Die Teammitglieder haben dadurch unterschiedliche Sichtweisen auf die Software, was häufig zu Missverständnissen unter den Teammitgliedern führen kann, die wiederum zu Fehlinterpretationen von Anforderungen und dadurch fehlerhafte Software führen.

Dieses Problem soll mit geeigneten Software Engineering-Methoden und -Ansätzen bewältigt werden. Dazu soll im folgenden der modellbasierte Softwareentwicklungsansatz, also die Verwendung von Modellen im Entwicklungsprozess, betrachtet werden.

Modelle haben den Vorteil, dass sie in einer formalen Sprache definiert werden und alle beteiligten Personen die gleiche Vorstellung von der Softwarearchitektur und -funktion besitzen. Zudem sind grafische Modelle leicht lesbar und verständlich.

Bereits im klassischen Wasserfallmodell (Kapitel 3), das bei der Softwareentwicklung in OS und Raumfahrtprojekten häufig zur Anwendung kommt, werden in den ersten Entwicklungsphasen Anforderungsdokumente und Unified Modeling Language (UML)-Diagramme definiert. Diese dienen jedoch oftmals nur zu Dokumentationszwecken und weichen während des Implementierungsprozesses immer mehr vom eigentlichen Code

ab, da Änderungen häufig aus Zeitmangel oder der höheren Codepriorität nur innerhalb des Codes umgesetzt werden. Diese Diagramme sollen nun mittels modellgetriebener Softwareentwicklung direkt in den Softwareentwicklungsprozess integriert werden.[5, S.3]

Dazu hat die Object Management Group (OMG) einen Entwicklungsansatz standardisiert, der aus einem formalen Modell den entsprechenden Systemcode generiert.[5, S.5] Dieses Konzept heißt Model Driven Architecture (MDA) und hat eine einheitliche und modellgetriebene Softwareentwicklung mit konsistenten, aufeinanderfolgenden Entwicklungsphasen zum Ziel.[6, S.1] Verschiedene Modelle unterschiedlicher Abstraktionsebenen werden entwickelt und in Form der Implementierung zusammengestellt.[5, S.5]

Mittels MDA soll die Spezifikation der Anwendungsfunktionalität von deren Realisierung auf einer bestimmten Softwareplattform getrennt werden.[6, S.2]

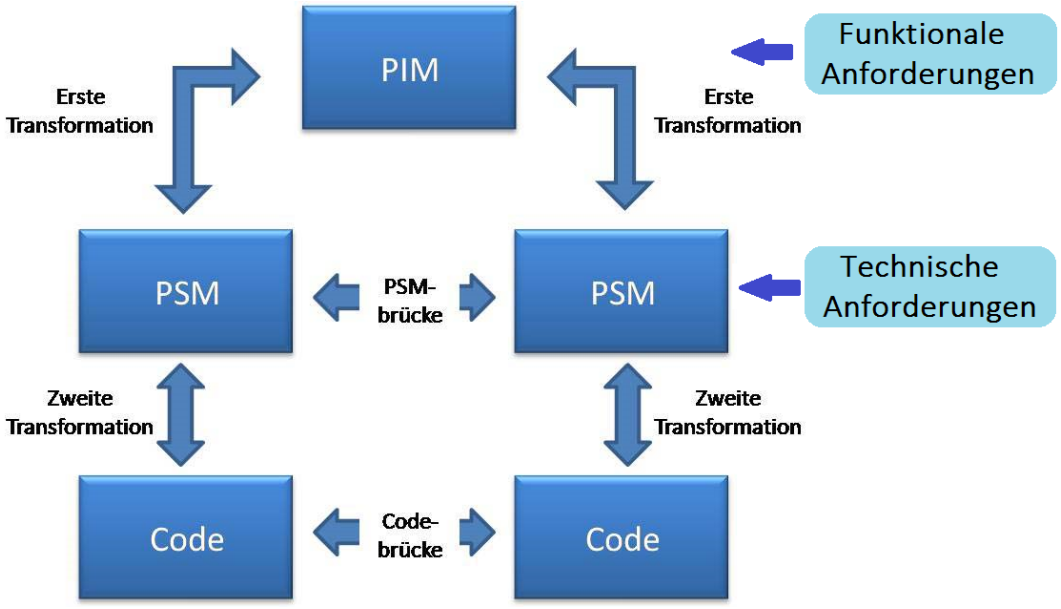


Abbildung 1: Entwicklungsprozess mit dem MDA-Ansatz [5, S.7]

Der Entwicklungsprozess (Abbildung 1) beim MDA-Ansatz beginnt mit der Definition eines plattformunabhängigen Modells (engl. Platform Independent Model (PIM)) beispielsweise in UML. Dieses Modell enthält die fachliche Spezifikation einer Software, ohne Beachtung der verwendeten Technologie.[6, S.5] Dies steigert die Portabilität der Software

und bietet zugleich eine Möglichkeit der Dokumentation, da das PIM eine Abbildung des Codes ist.[5, S.7]

Über eine oder mehrere Transformationen[1] können aus der PIM ein oder mehrere plattformabhängige Modelle (engl. Platform Specific Modells (PSM)) generiert werden. Änderungen am PIM werden durch Transformationen automatisch im PSM umgesetzt. Schließlich wird jedes generierte PSM in Code transformiert. Unterschiedliche PSM können über sogenannte Brücken miteinander kommunizieren, die bei der Transformation generiert wurden. Somit wird die Interoperabilität des Systems möglich.[5, S.7ff]

Durch diesen Entwicklungsprozess kann sich der Entwickler genauer auf die fachliche Spezifikation der Software konzentrieren, anstatt sich um deren Implementierung zu kümmern und Fehler im manuell erzeugten Programmcode zu suchen. Damit steigt die Produktivität der Softwareentwicklung.[5, S.7]

Die Modellierung des Softwaresystems beruht auf Abstraktion und Klassifizierung. Während durch Abstraktion Informationen, die zum Zeitpunkt der Softwareentwicklung irrelevant sind, versteckt werden, gruppiert die Klassifizierung Informationen mit gleichen Eigenschaften. Dadurch ist eine Wiederverwendung der Modelle zur Entwicklung ähnlicher Systeme möglich.[5, S.8]

Modelle werden in Modellierungssprachen (z. B. UML) verfasst. Diese Modellierungssprache definiert die Elemente, die in einem Modell vorhanden sind. Ein Metamodell der verwendeten Sprache definiert die Menge der Elemente, die in einem Modell verwendet werden.[5, S.9] Alle Entwickler können somit das Softwaremodell leichter nachvollziehen.

Zusammengefasst bietet die modellbasierte Softwareentwicklung folgende Vorteile:

- die Lesbarkeit der Funktionalität und Architektur wird insbesondere für Nicht-Entwickler erhöht, die Qualität verbessert und der Aufwand für die Wartung und Entwicklung mittel- und langfristig reduziert

[1] Eine Transformation enthält einen Satz an Spezifikationen, die eine Umwandlung des Quellmodells ins Zielmodell zulässt.[5, S.12]

- leichtere Zusammenarbeit für größere Entwicklerteams
- der Code entspricht weitgehend der Architektur und den Anforderungen
- die Wiederverwendbarkeit von Modellen und Code wird verbessert
- die Portabilität auf verschiedene Hardwareplattformen wird verbessert und der Aufwand reduziert
- die Notwendigkeit der Verfügbarkeit einer realen Hardwareplattform zur Integration und Test der Software wird minimiert
- die Einhaltung der Konsistenz zwischen Dokumentation und Software wird verbessert

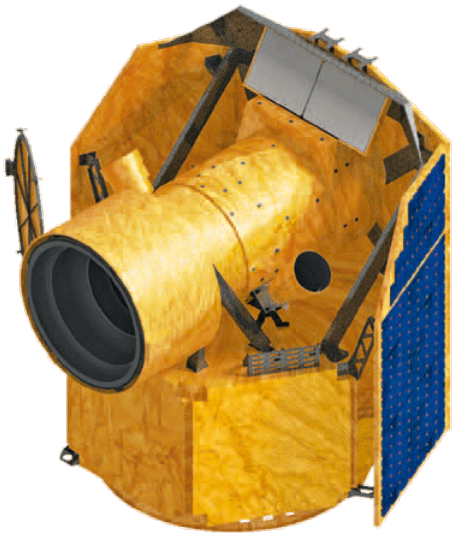
Demgegenüber sind folgende Nachteile zu nennen:

- der Einarbeitungsaufwand und der Aufwand der Erstellung eines ersten Modells bis zur lauffähigen Version ist höher gegenüber der klassischen Methode zur Softwareentwicklung
- Entwickler sind noch stärker auf eine funktionierende Werkzeugkette angewiesen
- ggf. fallen hohe Lizenzkosten bei der Nutzung ausgereifter kommerzieller Werkzeuge an
- generierter Verbindungscode ist oftmals schwer verständlich und kaum nachvollziehbar, individuelle Codeanpassungen durch den Entwickler werden schwierig

Einen modellbasierten Entwicklungsansatz wird von der TASTE-Technologie Architecture Analysis and Design Language (AADL) unterstützt. AADL ist ein von SAE International[2] entwickelter Standard und wurde zur Spezifikation, Analyse, automatischen Integration und Codegeneration von Echtzeit-Performance kritischen verteilten Systemen entwickelt. Eine Analyse des Systemdesigns vor der Systementwicklung und ein modellbasierter Entwicklungsansatz im gesamten Lebenszyklus des Systems wird somit unterstützt.[7]

[2] SAE International ist ein globaler Zusammenschluss von Wissenschaftlern, Ingenieuren und Fachleuten aus der Luftfahrt-, Automobil- und Nutzfahrzeugindustrie. http://de.sae.org/

3 On-Board Software des Projekts CHEOPS



http://cheops.unibe.ch/594/

Abbildung 2: Characterising Exoplanet Satellite

Als Beispielprojekt zur Entwicklung einer eingebetteten Flugsoftware wird das Projekt Characterising Exoplanet Satellite (CHEOPS), die erste Kleinsatellitenmission der ESA, näher betrachtet.

CHEOPS ist eine gemeinsame ESA-Schweiz Raumfahrtmission, die Ende 2017 startet. Das Projekt dient der Suche nach Planeten mittels extrem hoher Genauigkeitsphotometrie auf hellen Sternen, von denen bekannt ist, dass sie Planeten besitzen.

Teile der CHEOPS Software sollen beispielhaft mit TASTE modelliert werden, um die Nutzung der TASTE-Werkzeuge für ein reales Raumfahrtprojekt zu testen und zu bewerten. Dieses Kapitel gibt einen Überblick über die zu modellierende CHEOPS Software.

Das auf dem Satelliten CHEOPS integrierte Sensor Electronics Sub-system (SES) besteht aus zwei physisch voneinander getrennten Modulen: dem Focal Plane Module (FPM) und dem Sensor Electronics Module (SEM). Das SEM enthält die Sensor Control Unit (SCU) und die Power Conditioning Unit (PCU). Das SES wird durch die Back-End Electronics (BEE) über Power und Spacewire kontrolliert. Die SES On-Board Software

läuft auf der SCU und führt die geforderten Aufgaben der BEE aus.[8, S.10] Abbildung 3 zeigt das CHEOPS SES-BEE-Blockdiagramm.

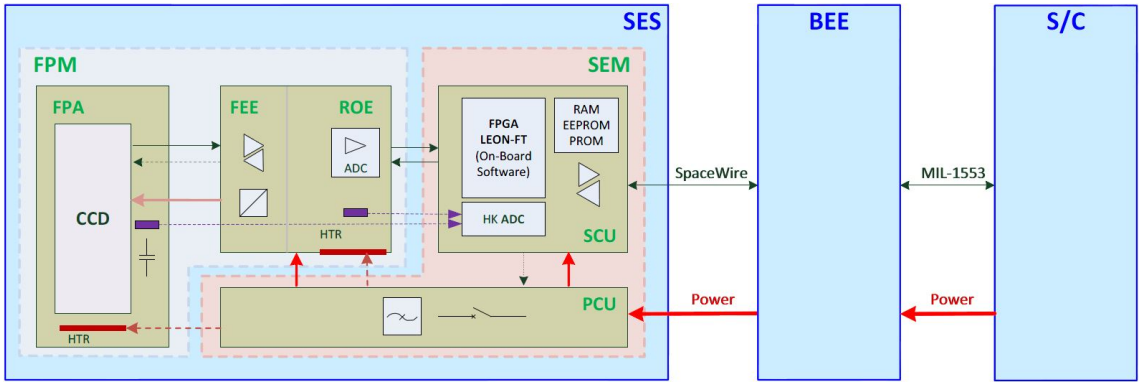


Abbildung 3: CHEOPS SES-BEE Block Diagram [8, S.10]

Die CHEOPS SES On-Board Software besteht aus zwei eigenständigen Teilen: der Primary Boot Software (PBS) und der Application Software (APS).[8, S.10]

Die PBS ist eine physisch separate, ausführbare Software, die eine Teilfunktionalität der APS, der aus Zuverlässigkeitsgründen realisiert und fest in einem PROM gespeichert ist. Der Fokus liegt dabei auf die minimale Funktionalität zum Management der Speicherung der APS im EEPROM und somit kleiner Codesize. Die PBS wird in dieser Arbeit nicht weiter betrachtet. Die Werkzeugsammlung TASTE würde lediglich für die APS eingesetzt werden, da bei der APS ein größerer EEPROM zur Code-Speicherung zur Verfügung steht.

Abbildung 4 zeit die Softwarearchitektur mit allen Komponenten der CHEOPS APS. Die APS soll als echtzeitfähige Software für Multiprozessorsysteme entwickelt werden. Auf der Betriebssystemebene befinden sich die Hardwaretreiber, die APS und Real Time Operating System (RTOS) Initialisierungskomponente sowie die Interrupt Service Routines (ISR). Die APS Initialisierungskomponente führt die Einstellungen der Hardware und des Echtzeitbetriebssystem (RTOS) durch. Die APS initialisiert alle Hardwareeinheiten und SES Teilsysteme und richtet diese ein.[8, S.49]

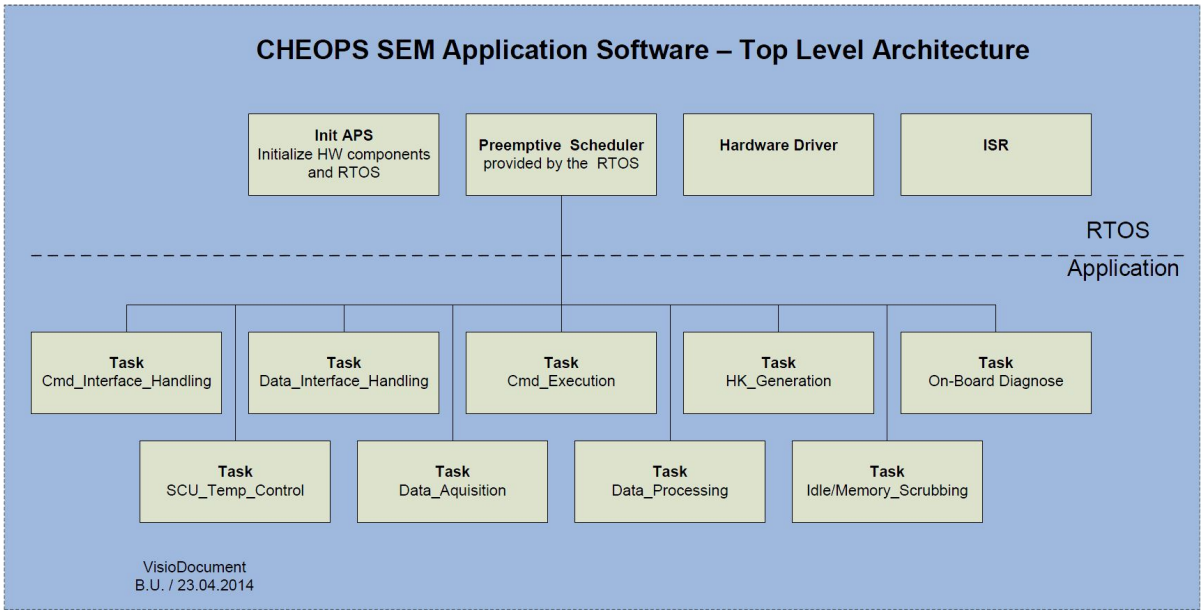


Abbildung 4: CHEOPS APS-Komponenten [8, S.49]

Auf der Anwendungsebene befinden sich die Ausführungstasks. Deren Aufgaben werden in Tabelle 1 beschrieben.

Die Telekommando- und Telemetriepakete, die durch diese Taks empfangen, verarbeitet und gesendet werden, folgen dem CCSDS/PUS-Standard. Der Aufbau dieser Pakete ist in den Anhängen A.1 bzw. A.2 nachzulesen.

Der ECSS–E–70–41A Standard (Siehe [9]) spezifiziert Standardservices des Protokolls zur Anwendung der Schnittstellenkommunikation sowie der Telekommando und Telemetriedaten Definition, die von den On-Board Anwendungsprozessen zur Erfüllung des Operationskonzeptes für Satellitenbeobachtung und -Kontrolle zur Verfügung gestellt werden. Jeder Service umfasst einen Satz verschiedener Funktionen.[9, S.13]

Jede dieser Funktionen wird mit einem speziellen Paket implementiert. Welches Paket zu welchem Service gehört, wird über den *Service Type* (*Packet Type*) im DataFieldHeader eines Paketes festgelegt. Der *Service Subtype* (*Packet Subtype*) identifiziert zusammen mit dem *Service Type* die implementierte Funktion des Paketes.[9, S.40]

APS-Task	Taskaufgabe
Memory_Scrubbing	wird als Leerlaufprozess des Schedulers ausgeführt
Cmd_Interface_Handling	Bearbeitung der Spacewire-Kommando-Schnittstelle: empfängt und prüft Kommandos von der BEE und speichert es in der Kommandoausführung-Queue; sofern angefordert wird ein Akzeptanzbericht erzeugt
Data_Interface_Handling	Bearbeitung der Spacewire-Daten-Schnittstelle: liest die Telemetriepakete aus der Empfangs-Queue und sendet die Pakete an die BEE
Cmd_Execution	liest alle Kommandos aus der Kommandoausführung-Queue, überprüft und führt diese aus; sofern angefordert wird ein Ausführungsbericht erzeugt
HK_Generation	erfasst die default und extended Housekeepingdaten und generiert Housekeepingdatenpakete für die BEE
On-Board-Diagnose	führt die On-Board Diagnose durch
SCU_Temp_Control	führt periodische Messungen der SCU/SES Temperaturen durch und ruft das Controller Unterprogramm auf
Data_Aquisition	konfiguriert die Generation der wissenschaftlichen Daten und startet oder triggert die Generation dieser Daten
Data_Processing	führt Front End Electronics (FEE) Datenverarbeitung und -formatierung durch und erzeugt Datenpakete zur Übertragung an an die BEE

Tabelle 1: CHEOPS APS-Tasks und deren Aufgaben [8, S.49-62]

Tabelle 2 listet alle Standardservices des ECSS–E–70–41A Standards mit Servicetyp und zugehörigem Servicenamen auf.

Ein Teil CHEOPS APS Komponenten soll mit den TASTE-Werkzeugen in einem Beispielsystem modelliert werden. Für das Beispielsystem sollen die drei Services, der *Telecommand verification service*, der *Housekeeping and diagnostic data reporting service* und der *Event reporting service*, beispielhaft mit den TASTE-Werkzeugen modelliert werden.

Service Type	Service Name
1	Telecommand verification service
2	Device command distribution service
3	Housekeeping and diagnostic data reporting service
4	Parameter statistics reporting service
5	Event reporting service
6	Memory management service
7	Not used
8	Function management service
9	Time management service
10	Not used
11	On-board operations scheduling service
12	On-board monitoring service
13	Large data transfer service
14	Packet forwarding control service
15	On-board storage and retrieval service
16	Not used
17	Test service
18	On-board operations procedure service
19	Event-action service

Tabelle 2: Standardservices des ECSS–E–70–41A Standards [9, S.50]

Mit den Funktionen des *Telecommand verification service* für das CHEOPS Projekt wird jedes empfangene Telekommandopaket verifiziert. Zunächst wird von der Zielanwendung geprüft, ob das empfangene Paket akzeptiert werden kann, d. h. vollständig und richtig ist. Es wird beispielsweise geprüft, ob der Service Typ oder Sub-Service Typ des Paketes unterstützt wird oder die Checksumme des Paketes richtig ist. Wird festgestellt, dass das Paket unbekannt oder fehlerhaft ist, wird ein Telemetriepaket für die Funktion *Telecommand Acceptance Report - Failure* mit dem entsprechenden Fehlercode zurück an die Benutzeranwendung am Boden gesendet. Ist das Paket bekannt und fehlerfrei, wird, sofern angefordert, ein Telemetriepaket der Funktion *Telecommand Acceptance Report - Success* zurück gesendet.[9, S.51-55]

Anschließend wird geprüft, ob das empfangene Kommando ausgeführt werden kann. Dazu wird beispielsweise geprüft, ob die Parameter in dem empfangenen Paket valide

sind und sich die CHEOPS On-Board Software im richtigen Modus befindet. Kann das empfangene Paket nicht ausgeführt werden, soll ein Fehlerbericht erzeugt werden. Ein Fehlerbericht wird mittels eines Telemetriepaketes der Funktion *Telecommand Execution Completed Report - Failure* mit entsprechendem Fehlercode zurück an die Benutzeranwendung am Boden gesendet. Ist eine fehlerfreie Ausführung des empfangenen Paktes möglich, wird ein Telemetriepaket mit der Funktion *Telecommand Execution Completed Report - Success* sofern gefordert zurück gesendet.[9, S.51-55]

Mit den Funktionen des *Housekeeping and diagnostic data reporting service* werden Informationen in Form von Telemetriepaketen und Statusinformationen vom Satelliten zur Benutzeranwendung am Boden geschickt. Für den Housekeeping Service gibt es zwei Berichtdefinitionen: den Bericht von *default* Housekeepingdaten sowie den Bericht der *extended* Housekeepingdaten. Jede Berichtdefinition besitzt ein Zeitintervall, das die Zeit angibt, in der die entsprechenden Housekeepingdaten erfasst werden.[9, S.61-63]

Der Housekeeping Service des CHEOPS Projektes umfasst fünf Funktionen. Davon sind drei Anforderungen an das Zielsystem, d. h. es sind Kommandos, die als Telekommandopakete implementiert werden. Die zwei verbleibenden Funktionen sind die eben genannten Berichtdefinitionen, die als Telemetriepakete implementiert werden.[9, S.66]

Die Funktion *Enable Housekeeping Parameter Report Generation* wird als Telekommandopaket implementiert und bestimmt, welche der Berichtdefinitionen aktiviert wird. Mit der Funktion *Disable Housekeeping Parameter Report Generation* wird angegeben, welche Berichtdefinition deaktiviert wird.[9, S.65] Eine weitere Anforderung an den Satelliten ist die Funktion *CMD_HK_Period*. Diese Funktion ist missionsspezifisch und gehört nicht zu den Standardservices des ECSS–E–70–41A Standards. Mit dieser Anfrage wird die Periode festgelegt, mit der eine oder beide Berichtdefinitionen ausgeführt werden sollen.

Die erfassten Daten der Berichtdefinitionen werden mit der Funktion *Housekeeping Parameter Report* an den Servicenutzer geschickt.[9, S.69]

Mit den Funktionen des *Event reporting service* werden ebenfalls Informationen, die von betrieblicher Bedeutung sind, zurück an die Benutzeranwendung am Boden geschickt. Zu diesen Informationen gehören Berichte über den Verlauf von Operationen und Aktivitäten, Berichte von Warnungen oder Fehlerberichte.[9, S.79]

Mit den Funktionen *Enable Event Report Generation* bzw. *Disable Event Report Generation* wird die Erzeugung der Ereignisberichte ein- bzw. ausgeschaltet.[9, S.80]

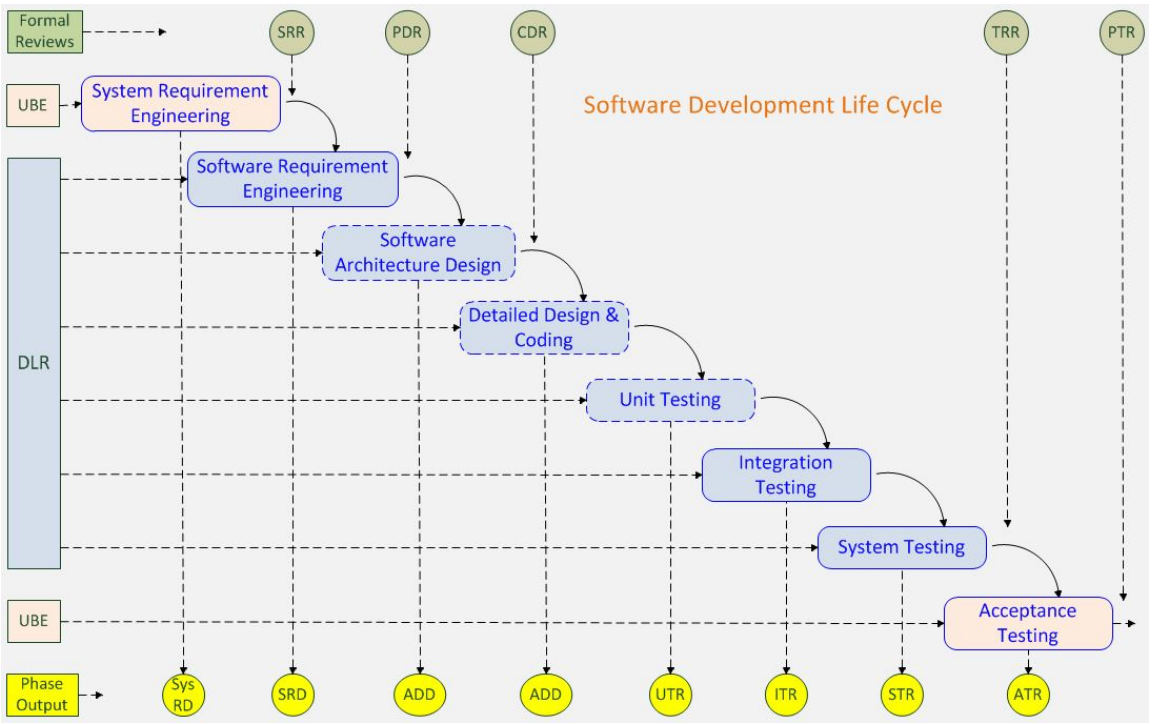
Um ein Telemetriepaket mit einem Verlaufsbericht zu senden, wird die Funktion *Normal/Progress Report* genutzt. Mit der Funktion *Error/Anomaly Report - Low Severity* wird ein Telemetriepaket mit einem Warnungsbericht gesendet. Die Funktion *Error/Anomaly Report - Medium Severity* sendet ein Telemetriepaket mit einem Fehlerbericht.[9, S.80]

Zur bisherigen Softwareentwicklung wurde das erweiterte Wasserfallmodell genutzt. Die Entwicklungsschritte *Software Requirement Engineering* bis *System Testing*, die in Abbildung 5 dargestellt sind, werden auf verschiedene Modellstufen des Systems (Development Model bis hin zum fertigen Flugmodell) angewandt oder je nach Bedarf innerhalb einer Entwicklungsphase wiederholt.

Für jeden Entwicklungsschritt werden Dokumente als Ausgabe erzeugt. Diese enthalten Anforderungen, technische Spezifikationen, Testberichte und Softwaredokumentationen für den Gebrauch und Instandhaltung.[10, S.12ff]

In den Phasen *Software Architecture Design* und *Detailled Design & Coding* könnte die Modellierung und Codegenerierung des Systems mit den TASTE-Werkzeugen erfolgen. In den Phasen *Unit Testing* und *Integration Testing* könnte das Testen mit den TASTE-Werkzeugen stattfinden.

Die SES On-Board Software (PBS und APS) wird hauptsächlich in C programmiert. Die Zielplattform ist ein LEON3FT Prozessor. Die Entwicklung erfolgt auf einem PC mit Werkzeugen zum cross-kompilieren und cross-profilieren. Tests der SES On-Board Software finden auf der Zielplattform statt.[10, S.14]



UBE = Universität Bern Ergebnisse: Sys RD = CHEOPS System Requirement Document, SRD = Software Requirement Document, ADD 0 Software Architecture Design document, UTR = Unit Test Report, ITR = Integration Test Report, STR = System Test Report, ATR = Acceptance Test Report
Formale Reviews: SRR = Software System Requirements Review, PDR = Preliminary Design Review, CDR = Critical Design review, TRR = Test Readiness Review, PTR = Post Test Review

Abbildung 5: CHEOPS Software Development Life Cycle [10, S.13]

Mit dieser Arbeit wird geprüft, ob die Entwicklungsschritte *Software Architekture Design* bis *Integration Testing*/*System Testing* mit den TASTE-Entwicklungswerkzeugen übernommen werden könnten.

4 TASTE Entwicklungswerkzeuge

Das folgende Kapitel gibt einen Überblick über die TASTE-Entwicklungswerkzeuge und den Entwicklungsprozess einer Software mit den TASTE-Werkzeugen.

Da abgesehen von den angebotenen TASTE-Werkzeuge auch externe Werkzeuge zur Softwareentwicklung mit TASTE genutzt werden können [4, S.29], werden im Anschluss einige dieser nicht zur TASTE-Werkzeugsammlung gehörenden Werkzeuge erläutert.

Es gibt verschiedene Entwicklungswerkzeuge, die ebenfalls einen MDA-Ansatz verfolgen und einige Anforderungen an die Entwicklung von Raumfahrtsoftware erfüllen. Zwei solcher modellbasierten Entwicklungswerkzeuge, die als Alternative zu den TASTE-Werkzeugen verwendet werden können, werden am Ende dieses Kapitels aufgeführt.

4.1 Konzept der TASTE Entwicklungswerkzeuge

TASTE ist eine Werkzeugsammlung zur Modellierung und Entwicklung heterogener, eingebetteter Systeme. Die Werkzeugsammlung wird seit 2008 von verschiedenen Personen und Unternehmen als freie, open-source Software entwickelt.[11, S.1]

TASTE wird ursprünglich zur Entwicklung von klein bis mittelgroßen Systemen in der Robotik und Regelungssystemen eingesetzt. Jedoch dient TASTE zurzeit nur zum Experimentieren mit neuen Softwaretechnologien und zu Bildungszwecken, da TASTE noch in der Entwicklungsphase ist.[11, S.10]

Die Werkzeuge sind in einer virtuellen Maschine[1] zusammengetragen und somit direkt einsatzfähig verfügbar. Die virtuelle Maschine umfasst eine volle Debian Linux Installation und einen Update-Mechanismus, um immer mit der aktuellsten TASTE-Version arbeiten zu können. Eine manuelle Installation ist ebenfalls möglich. Anleitungen dafür sind unter http://taste.tuxfamily.org/wiki/index.php?title=Manual_installation_on_a_native_platform zu finden. Diese beziehen sich auf die Installation in einem Linux System. Ein Windows GUI Prototyp existiert bereits und eine Anleitung

[1] http://download.tuxfamily.org/taste/taste-vm.tar.gz

zur Installation befindet sich unter http://taste.tuxfamily.org/wiki/index.php?title=TASTE_GUI_for_Windows.[12] Zur Bearbeitung der Aufgabe wurde die bereitgestellte virtuelle Maschine verwendet.

Einen Überblick über die Modellierungswerkzeuge, die zur Umsetzung des TASTE-Prozess genutzt werden, gibt Abbildung 6. Die Vorstellung des TASTE-Prozesses erfolgt im Anschluss.

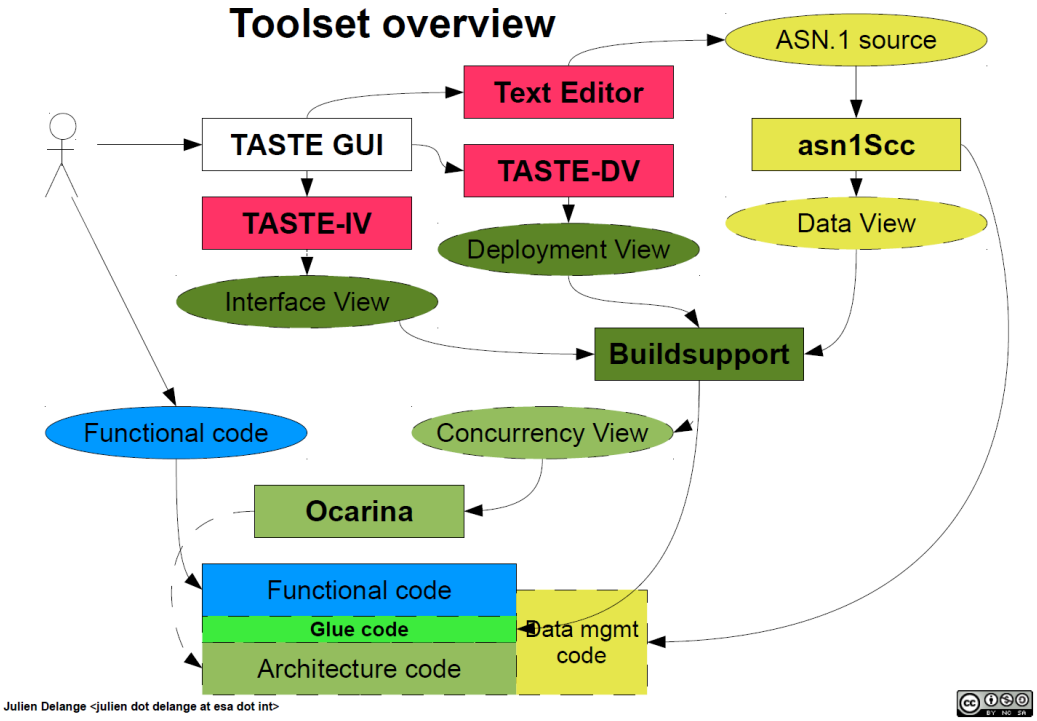


Abbildung 6: Überblick der TASTE-Werkzeugsammlung [13, S.20]

Ziel von TASTE ist es, eingebettete Echtzeitsysteme zu entwickeln, die durch den Aufbau korrekt sind (engl. „correct by construction“). Der Entwickler beschreibt die funktionalen Ansätze des Systems auf höchster Ebene und deren Zuweisung zur Hardware. TASTE generiert daraus automatisch Code, der das System implementiert und das Systemmodell nahezu identisch repräsentiert.[4, S.19]

TASTE benutzt mehrere Schlüsseltechnologien wie ASN.1 und AADL zur Erfassung der Daten und der Systemarchitektur, Codegeneratoren, Echtzeitbetriebssysteme, etc.[4, S.9]

TASTE umfasst verschiedene Modellierungswerkzeuge, die unterschiedlichen Code mit spezifischen Datenstrukturen generieren. Diese verschiedenen Datenstrukturen zusammenzufügen, wäre sehr zeitaufwendig und fehleranfällig, da es sich dabei um low-level Details handelt. TASTE übernimmt diese Aufgabe automatisch und garantiert, dass beim „zusammenkleben“ (engl. „glue-ing“) der funktionalen Komponenten keine Fehler auftreten.[4, S.9ff]

Für den Datenaustausch zwischen den verschiedenen Modellierungswerkzeugen und Sprachen werden die vom ASN1SCC generierten Encoder- und Decoder-Funktionen genutzt. Dieser Prozess ist vollautomatisiert und wird durch die TASTE Data Modeling Toolchain (DMT) von SEMANTIX durchgeführt.[4, S.10]

Mit TASTE können Anwendungen für verschiedene Architekturen, wie x86 mit Linux, Mac OS, RTEMS oder ARM mit RTEMS und Linux oder SPARC (LEON) mit RTEMS und OpenRavenscar, entwickelt werden. Während der Entwicklung eines Systems kann die Verteilung der Systemfunktionen auf eine Architektur leicht verändert werden. Beispielsweise können Funktionen zu einzelnen Prozessoren hinzugefügt werden. Dies wird durch die Trennung des Systemdesigns in Data View, Interface View und Deployment View ermöglicht.[4, S.10]

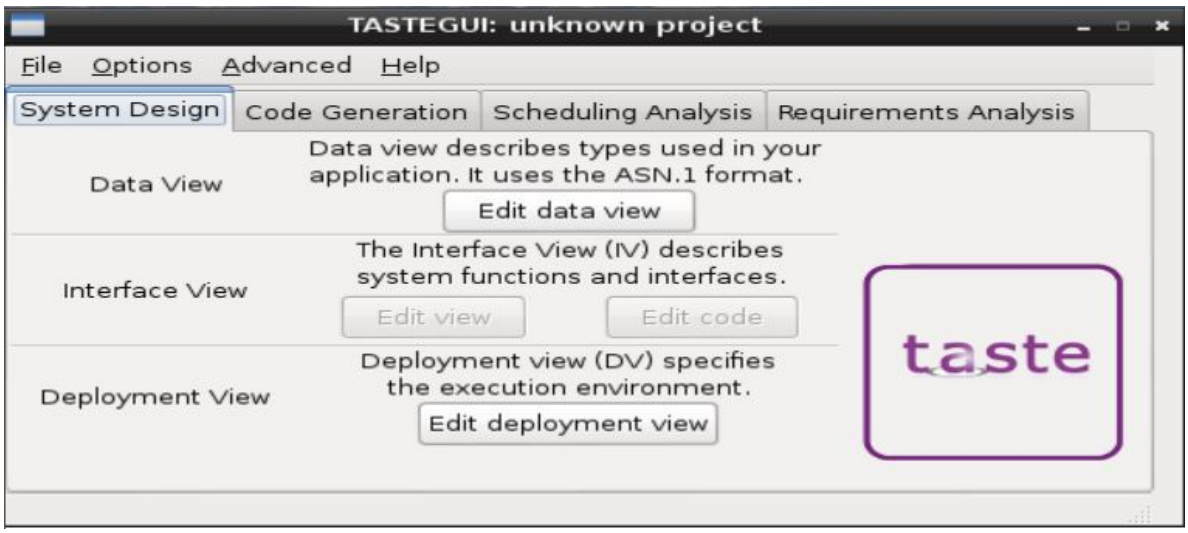


Abbildung 7: TASTEGUI

TASTE bietet eine eine grafische Schnittstelle TASTEGUI (Abbildung 7), um den Benutzer beim Entwicklungsprozess eines Systems zu unterstützen. Die TASTEGUI ist ein Werkzeug, dass von der ESA entwickelt wurde.[4, S.28,S.14] Alternativ zur TASTEGUI kann der Entwicklungsprozess über spezielle Konsolenbefehle der TASTE-Werkzeugen durchgeführt werden.

Der gesamte TASTE-Prozess zur Entwicklung eines Systems kann in sechs Schritte zusammengefasst werden[12, S.11]:

1. Beschreibung der logischen Systemarchitektur und -schnittstellen mit AADL und ASN.1

Im Data View wird das Datenmodell für den Nachrichtenaustausch zwischen den Teilsystemen manuell spezifiziert. Dabei werden die Typen und Einschränkungen der auszutauschenden Nachrichten mittels ASN.1 festgelegt und mit den ASN1SCC in Encoder- und Decoder-Funktionen der Sprache C oder Ada übersetzt. Die ASN.1-Nachrichtendefinitionen werden automatisch in AADL-Modelle übertragen.[4, S.17ff]

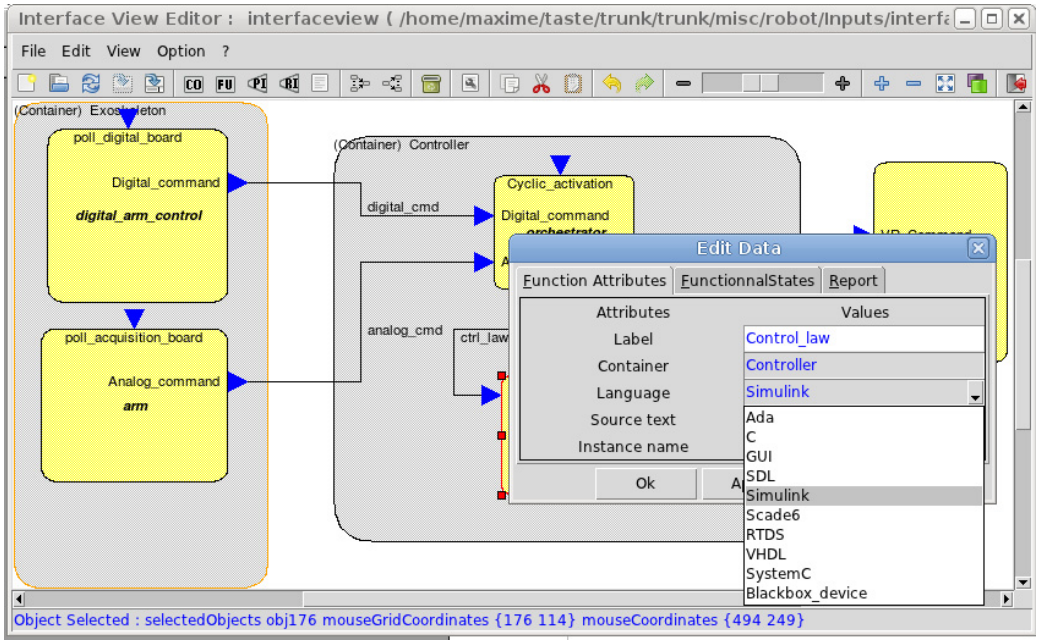


Abbildung 8: Oberfläche des TASTE-Interface Views [11, S.3]

Die definierten Datentypen werden anschließend im Interface View (Abbildung 8) zur Beschreibung der Schnittstellen genutzt.[4, S.18] Der Interface View dient zur Beschreibung der logischen Systemarchitektur mittels Container und Funktionen.[12, S.16] Es wird die grafische Repräsentation der standardisierten Textsprache AADL genutzt.[11, S.3] Zur Definition der Systemfunktionen gehört die Spezifikation der Funktionseigenschaften wie die Ausführungssprache, Periode, Kontextparameter, etc.[13, S.13]

Die wichtigste Definition im Interface View ist die Spezifikation der Schnittstellen zur Kommunikation mit anderen Funktionen. Dabei werden zwei Schnittstellenarten unterschieden: Provided Interface (PI) und Required Interface (RI). Das PI ist ein Service, der von einer Funktion angeboten wird. Das RI legt fest, dass eine Funktion mit einer anderen Funktion verbunden werden muss. Ziel ist es, damit Funktionen zu verbinden und die Abhängigkeiten und Interaktionen zu zeigen.[12, S.20]

Für die Ausführungssprache einer Funktion kann der Entwickler verschiedene Sprachen, die von TASTE unterstützt werden, wie C, Ada, SDL, Simulink, etc. in den Eigenschaften der Funktion angeben. Die Kommunikation der unterschiedlichen Funktionen erfolgt zum einen über automatisch generierte ASN.1 Encoder- und Decoder-Funktionen, die die Schnittstellenparameter ordnen sowie zum anderen über automatisch generierte PolyORB-Hi-Kontainer[2], die die Kommunikation der Objekte (Tasks, Threads) instanziiert.[4, S.18]

2. Generierung von Code-Gerüsten und Schreiben des geeigneten Codes oder Modelle

Nachdem alle Eigenschaften und Anforderungen der Systemfunktionen spezifiziert wurden, generiert TASTE für jede definierte Funktion Code-Gerüste in der jeweiligen Ausführungssprache der Funktion. Diese müssen durch den Entwickler mit der gewünschten Funktionalität befüllt werden. Die Schnittstellen der Systemfunktionen werden komplett automatisch generiert.[4, S.18]

[2] PolyORB-Hi ist die Middleware, die den generierten AADL-Code mit dem RTOS verbindet.[4, S.26]

3. Erfassung der Systemhardware und -aufstellung

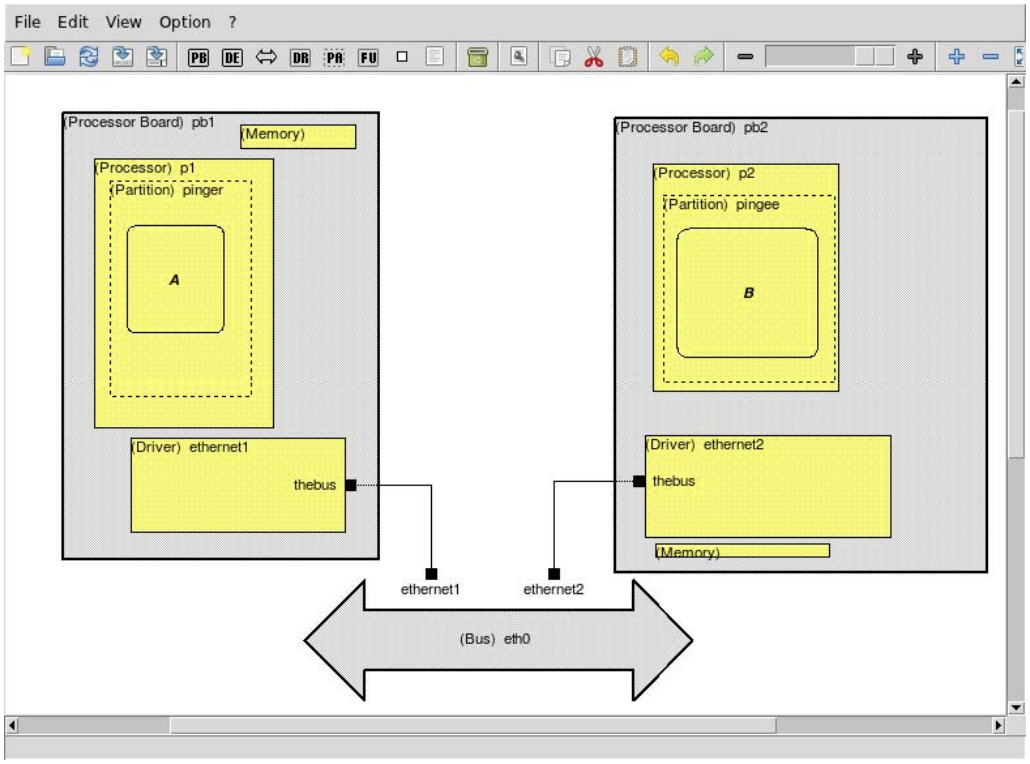


Abbildung 9: Oberfläche des TASTE-Deployment Views [11, S.4]

Im Deployment View (Abbildung 9) wird die Ausführungsumgebung und somit die verteilte Architektur mit allen Anforderungen erfasst, das heißt verwendete Prozessoren, Busse und Treiber. Die Funktionen aus dem Interface View werden im Deployment View den Hardwarekomponenten (Prozessoren) zugewiesen.[13, S.17ff] Ein Bus spezifiziert die Interprozessorkommunikation bei verteilten Systemen. Treiber werden zur Buskontrolle benötigt.[11, S.4] Damit wird eine Repräsentation des Systems auf höchster Ebene möglich. Schließlich wird ein AADL-Modell mit Hardwarekomponenten erzeugt. [13, S.23]

4. Modellverifikation

Das von der ESA entwickelte Buildsupport-Tool generiert automatisch aus dem Interface View und dem Deployment View den Concurrency View sowie Glue Code (Verbin-

dungscode) zwischen dem Architektur- und dem Anwendungscode. Das Ergebnis ist ein AADL-Modell mit Software- und Hardwarekomponenten [13, S.25], das anschließend von Ocarina[3] verarbeitet wird, um das System in C oder Ada zu erzeugen.[4, S.18]

Der Concurrency View ist eine Systembeschreibung, die Tasks, Daten und Interprozess-Verbindungen beinhaltet.[4, S.21]

Der ConcurrencyView-Editor stellt Funktionen zur Schedulability-Analyse und Realisierbarkeit des Systems zur Verfügung. Mit diesen Funktionen kann geprüft werden, ob Deadlines von Tasks eingehalten werden. Zudem kann das Verhalten eines Systems in Hinblick auf mögliche Probleme untersucht werden.[4, S.43]

Um die Schedulability-Realisierung zu bewerten, wird im ConcurrencyView-Editor Cheddar als Scheduling-Analysewerkzeug angeboten. Die Cheddar Ausgabe basiert auf Scheduling-Theorien und Realisierungstests.[4, S.43ff]

Mit dem im ConcurrencyView-Editor enthaltenen Marzhin Scheduling-Simulator (Abbildung 10) wird die Simulation der Ausführung jedes Tasks (läuft, wartet auf eine Ressource, schläft, etc.) sowie der Status geteilter Daten (gelockt, freigegeben, etc.) gezeigt.[4, S.44]

Nähere Informationen zu Marzhin können in [4] nachgelesen werden.

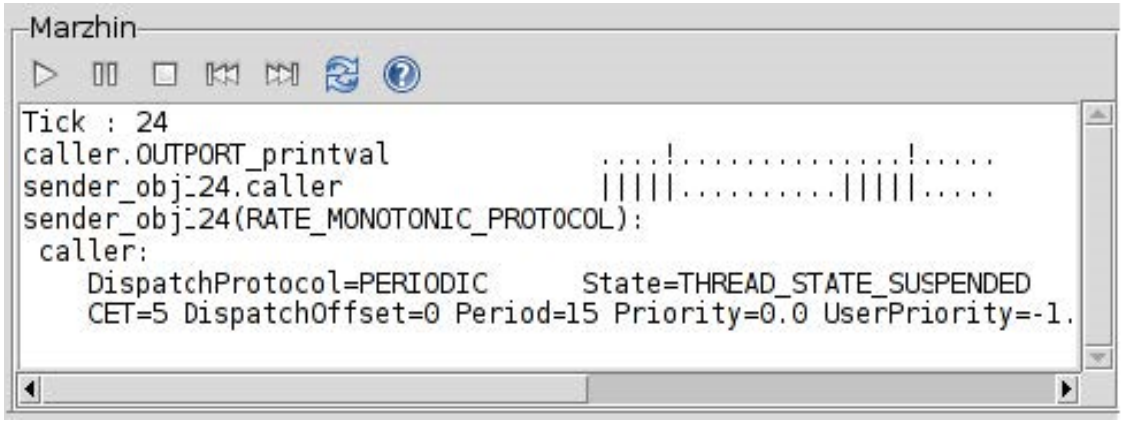


Abbildung 10: Marzhin Simulator [14, S.5]

[3] Ocarina ist ein Werkzeug zur Manipulation von AADL Modellen und bietet Codegenerationseigenschaften an, die Code für Echtzeitmiddleware wie PolyORB erzeugen.[4, S.25]

MAST ist neben Cheaddar ebenfalls ein Schedulability-Analysewerkzeug, das von der grafischen Schnittstelle der TASTE-Entwicklungswerkzeuge TASTEGUI angeboten wird. MAST ist eine modellbasierte Werkzeugsammlung, dass anhand von Eingangsmodellen das Echtzeitverhalten des Systems analysiert.[4, S.135] Genauere Details zu den MAST-Werkzeugen und deren Features können in [15] nachgelesen werden.

Nähere Informationen zur Nutzung von MAST innerhalb der TASTE-Werkzeuge befinden sich in [4] Kapitel 16.

5. Erstellung des Systems und Herunterladen auf das Ziel

Der Build-Prozess wird mittels Orchestrator[4] automatisiert. Aus dem Data View, dem Interface View, dem Deployment View und dem funktionsgemäßen Code wird letztendlich eine Binärdatei erzeugt, indem die verschiedenen Werkzeuge z.B. das Buildsupport-Tool, Ocarina, etc. aufgerufen werden.[4, S.27]

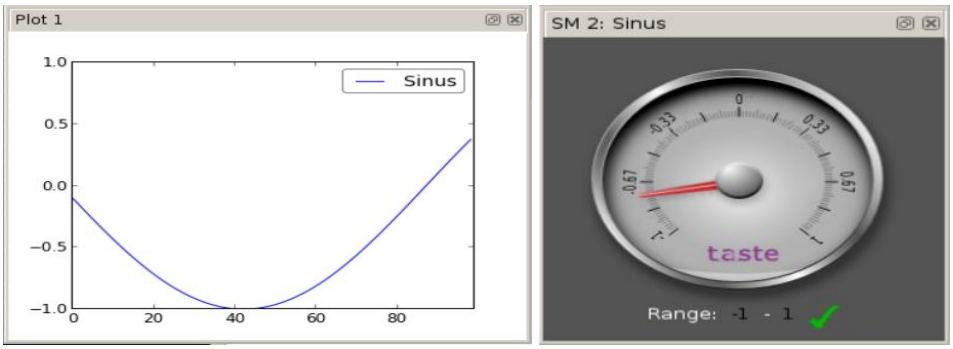
Orchestrator bietet die Möglichkeit mit gcov die Anweisungsüberdeckung der generierten Binärdatei zu überprüfen und mit gprof eine Performanceanalyse des Systems durchzuführen.[4, S.85ff]

6. Überwachung und Interaktion mit dem System zur Laufzeit

Zur Kommunikation mit dem zu entwickelnden eingebetteten System besteht die Möglichkeit eine vollständige GUI zu definieren, die automatisch generiert wird. Die GUI ermöglicht eine interaktive Echtzeitüberwachung und Steuerung des Systems. Eine grafische Überwachung der Telemetriedaten, wird durch GnuPlot (Abbildung 11) ermöglicht.[4, S.11]

Mit den tracer.py und tracerd.py Programmen kann der Nachrichtenaustausch der automatisch generierten GUI mit dem laufenden System in Echtzeit über den frei verfügbaren PragmaDev MSC Tracer kontrolliert werden.[4, S.11] Abbildung 12 zeigt, wie die Echtzeitüberwachung des Nachrichtenaustausches mit dem MSC Tracer aussieht.

[4] Orchestrator ist ein Programm zur Automatisierung des Build-Prozesses und erzeugt eine Binärdatei, die der Systemdurchführung entspricht.[4, S.27]



(a) Grafische Überwachung im Diagramm (b) Grafische Überwachung mit dem Meter

Abbildung 11: Grafische Überwachung der Telemetriedaten in Echtzeit

Mit dem tracer.py Werkzeug wird der Austausch der Telekommando- und Telemetriepakete in einer Message Sequence Chart (MSC) Datei gespeichert. Diese aufgezeichneten MSC Daten werden mit dem msc2py Werkzeug in ein Python Script konvertiert. Das Script führt das Szenario zur Laufzeit aus.[4, S.13]

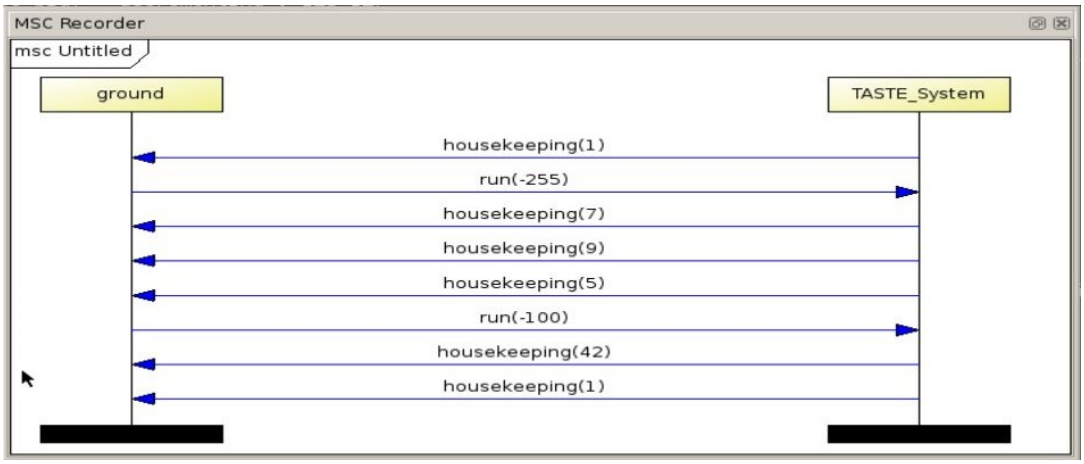


Abbildung 12: Überwachung der Telekommando- und Telemetriepakete mit MSC Tracer

Die TASTE Werkzeuge erzeugen automatisch Python-Brücken, die Zugriff auf die ASN.1 Parameter und direkten Zugang zu den Telekommando- und Telemetriepaketen des Systems zur Laufzeit haben. Der Benutzer kann somit mit Python Scripten Regressionstests durchführen.[4, S.13]

4.2 Ergänzende Werkzeuge

Wie bereits angedeutet, können neben den internen TASTE-Werkzeugen externe Werkzeuge zusammen mit den TASTE-Werkzeugen genutzt werden. Dieses Kapitel wird auf zwei Werkzeuge, die nicht zu der TASTE-Werkzeugsammlung gehören eingehen, die anstelle interner Werkzeuge verwendet werden können und auf die die Entwickler für eine professionelle Systementwicklung verweisen.

Pragmadev RTDS

Real Time Developer Studio (RTDS) ist ein von Pragmadev entwickeltes Modellierungswerkzeug, das für Kommunikationssysteme und Echtzeitanwendungen entwickelt wurde. Es umfasst Anforderungen, Spezifikationen, Prototypentwicklung, Design, Debuggen auf dem Target und Tests. Die Hersteller versprechen eine bessere Qualität, eine verringerte Entwicklungszeit, eine selbst dokumentierte und von einem Echtzeitbetriebssystem zum anderen portierbare Anwendung, die Fähigkeit der Modellüberprüfung sowie Informationen zur Nachvollziehbarkeit.[16]

RTDS ist ein grafischer Editor mit Kontexthilfe und Syntaxüberprüfung, um ein konsistentes System zu erstellen.[17, S.3]

RTDS kann anstelle des SDL-Editors OpenGEODE verwendet werden. Die TASTE-Entwickler empfehlen, zur Entwicklung professioneller Anwendungen RTDS zusammen mit TASTE zu benutzen, da RTDS im Gegensatz zu OpenGEODE eine bessere Funktionalität besitzt. OpenGEODE befindet sich noch in der Entwicklung und dient laut den Entwicklern als Technologiedemonstrant.[18] Wie die Nutzung des RTDS-Werkzeugs zusammen mit TASTE realisiert wird, kann in [4] nachgelesen werden. Dieses Werkzeug wurde im Rahmen dieser Arbeit nicht getestet, da die Abteilung IOS keine Lizenz für dieses Werkzeug besitzt.

Weitere Informationen zum RTDS-Werkzeug und seinen angebotenen Funktionen können in [17] nachgelesen werden.

Rapitime

Rapita Verification Service (RVS) ist eine Werkzeugsammlung zur Verifikation von Systemen, die u. a. Rapitime umfasst und in TASTE integriert werden soll. Diese Werkzeugsammlung wird bereits in der Luftfahrt-, der Automobil- und der Bahnindustrie zu Verifikations- und Zertifizierungszwecken eingesetzt.[19, S.1]

RVS führt zum einen eine Strukturanalyse des Quellcodes aus. Diese basiert auf der Vorverarbeitungsfähigkeit des Prozessors. An dieser Stelle sind die möglichen Ausführungspfade identifiziert und sind typische Punkte zur Instrumentierung des Codes.[19, S.1]

Zum anderen führt RVS eine dynamische Analyse der ausführbaren Binärdatei aus, die aus dem zuvor analysierten Quellcode erzeugt wurde. Informationen, die bei den Ausführungen mittels der Instrumentierungspunkte erhalten wurden, werden bearbeitet und mit den zuvor erfassten Strukturinformationen verbunden. Das Ergebnis dieser Analyse sind Ausführungszeiten, Ergebnisse der Testüberdeckung, Worst Case Execution Time (WCET), etc.[19, S.1]

Rapitime ist ein möglicher Ersatz für die Testfunktion der TASTEGUI. Nähere Informationen zu Rapitime befinden sich in [19]. Das Rapitime-Werkzeug wurde nicht getestet, da die Abteilung IOS keine Lizenz für dieses Werkzeug besitzt.

4.3 Alternative Modellierungswerkzeuge

Neben der TASTE-Werkzeugsammlung gibt weitere Entwicklungswerkzeuge, die einen modellgetriebenen Softwareentwicklungsansatz verfolgen und eventuell zur Entwicklung von Raumfahrtsoftware geeignet sind. Zu Informationszwecken werden im folgenden zwei Werkzeuge vorgestellt, die als Alternative zu TASTE genutzt werden können.

IBM Rational Rhapsody

IBM Rational Rhapsody ist ein proprietäres, in allen Industrien weit verbreitetes Modellierungswerkzeug für eingebettete Software.[20] Es unterstützt den Entwickler bei dem

Entwurf, Test und Dokumentation der Software.[21] Mit Rhapsody können Softwaremodelle beispielsweise in SysML[5] oder UML erstellt werden.[20]

Rhapsody bietet folgende Funktionen:

- Simulation der Modelle
- Automatische Codegenerierung
- Reverse Engineering
- RoundTrip Engineering
- Effiziente Bedienung
- Testautomatisierung

Rhapsody wird in verschiedenen Editionen angeboten.[20] Zur Entwicklung eingebetteter Echtzeitsysteme wird von den Entwicklern Rational Rhapsody Developer empfohlen.[22]

Rational Rhapsody Developer bietet eine visuelle Entwicklungsumgebung zur Generierung von C/C++, Java und Ada Code. Durch simulations- und modellbasierte Tests ist eine frühe Designvalidierung möglich. Zur Rückverfolgung von Anforderungen werden die Anforderungen der Designelemente und Testfälle gespeichert.[22]

Weitere Informationen zu Rhapsody sind unter http://www-01.ibm.com/support/knowledgecenter/#!/SSB2MU_8.0.0/com.ibm.rhp.overview.doc/topics/rhp_c_po_rr_product_overview.html zu finden.

Wie auch mit den TASTE-Entwicklungswerkzeugen kann mit Rhapsody in Simulink oder SDL modelliert werden.

Während verschiedene Features, z. B. die Unterstützung der Teamarbeit, Lebenszyklus-Unterstützung, etc., durch den Einsatz unterschiedlicher Rational Produkte möglich sind, werden Technologien wie ASN.1 und AADL nicht verwendet.[21]

[5] SysML, eine Abwandlung von UML, ist eine visuelle Modellierungssprache zur Entwicklung verschiedener komplexer Systeme. http://sysml.org/

Telelogic TAU Generation2

Telelogic TAU Generation2 ist ein Werkzeug zur Entwicklung von Echtzeitsystemen.

Es bietet eine gemeinsame Entwicklungsumgebung für Projektmanager, Systemanalytiker, Softwaredesigner, Programmierer, Tester, etc. TAU Generation2 besitzt eine fortgeschrittene visuelle Modellierungsumgebung, in der die Systemarchitektur, das Design und das Verhalten einer Software mittels UML 2.0 auf einem hohen Level zu spezifizieren. Die modellgetriebene Entwicklung erlaubt dem Entwickler sich auf die Architektur und das Design der Software zu konzentrieren und muss sich nicht auf Implementationsdetails fokussieren.[23, S.4]

Des weiteren bietet das Werkzeug die Möglichkeit, das System bereits in einer frühen Entwicklungsphase zu testen. Das Systemverhalten kann analysiert und korrigiert sowie Fehler frühzeitig entdeckt und entfernt werden.[23, S.6]

Für eingebettete Systeme kann TAU Generation2 aus dem visuellen Softwaredesign automatisch ausführbaren C Code generieren. Neben C werden Sprachen wie C++ und Java unterstützt.[23, S.9]

Automatische System- und Integrationstest werden für das System durchgeführt. Es bietet ein Werkzeug als Metrik-Tool und zur Softwarequalitätssicherung.[23, S.8ff]

TAU Generation2 unterstützt so wie die TASTE-Werkzeuge Technologien wie SDL und MSC.[23, S.8] ASN.1 und AADL werden auch bei dieser Alternative nicht verwendet.

Genauere Informationen zu TAU Generation2 und dessen Funktionalitäten können in [23] nachgelesen werden.

Im Rahmen der Aufgabenstellung wird nicht weiter auf alternative Werkzeuge zu den TASTE-Werkzeugen eingegangen, da es Ziel der Arbeit ist, die TASTE-Werkzeuge zu testen und zu bewerten. Im folgenden wird sich die Arbeit auf die Softwareentwicklung mit den TASTE-Werkzeugen konzentrieren.

5 Modellierung der Schnittstellenkommunikation

Zur Modellierung von Software mit den TASTE-Entwicklungswerkzeugen muss zunächst das Datenmodell spezifiziert werden. Dieses enthält alle Typen und Einschränkungen der auszutauschenden Nachrichten im System. Die Spezifizierung des Datenmodells erfolgt mit der ASN.1 Notation. Einen Überblick über die ASN.1 Notation bietet das Kapitel 5.1.

Die spezifizierten Datentypen werden bei der Modellierung der Systemfunktionen und deren Schnittstellen genutzt.

Das Kapitel 5.2 beschreibt die Definition des Datenmodells für das Beispielprojekt und dessen Nutzung bei der Modellierung eines Teils der CHEOPS APS auf höchster Ebene.

5.1 Überblick der ASN.1 Notation

Abstract Syntax Notation One (ASN.1) dient zur abstrakten Definition von Datenstrukturen für Nachrichten, z. B. im Bereich der Telekommunikation. Es ist ein ursprünglich in den 1980-er Jahre entwickelter ISO[1]/IEC[2] und ITU-T[3] Standard.[24, S.9]

ASN.1 wird zu Beschreibung der strukturellen Aspekte der auszutauschenden Daten genutzt. Es ist unabhängig von der verwendeten Programmiersprache, der Hardwareplattform, dem verwendeten Betriebssystem, der Verschlüsselungsmethode sowie von der physikalischen Darstellung der Daten. ASN.1 ist für Nachrichtenbeschreibungen von einfachen und komplexen Datenübertragungen geeignet.[24, S.9]

Mit ASN.1 können Kommunikationsprotokolle zwischen entfernten, heterogenen Systemen zur Vereinfachung der Kommunikation spezifiziert werden. Dafür stellt ASN.1 grundlegende Typen wie INTEGER, BOOLEAN, CHARACTER STRING, BIT STRING, etc. zur Beschreibung der logischen Datenstruktur in den Nachrichten zur Verfügung.

[1] International Standards Organization (ISO) ist eine internationale Organisation zur Entwicklung internationaler Standardnormen im Technologiebereich.[24, S.9]

[2] Internationale Elektrotechnische Kommission (IEC) ist eine internationale Organisation für internationale Standards im Elektrotechnik- und Elektronikbereich.[24, S.9]

[3] International Telecommunication Union (ITU)'s Telecommunication Standardization Sector (ITU-T) entwickeln Standards im Bereich der Telekommunikation.[24, S.9]

Zusätzlich zu den Basistypen können zusammengesetzte Typen wie Strukturen, Listen, etc. definiert werden. Mit ASN.1 können für jeden definierten Datentyp Einschränkungen, beispielsweise die Größe eines Typs, festgelegt werden.[24, S.9ff]

Der Programmierer muss bei der Nachrichtenbeschreibung mit ASN.1 nicht auf die interne Darstellung des Systems eingehen, wenn er die Encoder- und Decoder-Funktionen für den Nachrichtenaustausch nutzt, die automatisch von einem ASN.1 Compiler in einer beliebigen Programmiersprache generiert werden.[24, S.11]

Ein ASN.1 Compiler, der von den TASTE-Entwicklungswerkzeugen genutzt wird, ist der ASN.1 Space Certifiable Copmiler (ASN1SCC). Der ASN1SCC ist ein open-source Compiler, der zur Erfüllungen der Anforderungen an Software der Raumfahrt entwickelt wurde. Der Compiler generiert automatisch aus einer ASN.1-Spezifikation Encoder- und Decoder-Funktionen in einer beliebigen Zielsprache zur Kommunikation heterogener Systeme, die diesen Anforderungen gerecht werden.[24, S.12ff]

Der ASN1SCC bietet zusätzliche Features an. Zum Beispiel eine Testfunktion zur automatischen Generierung von Unittests für die erzeugten Encoder- und Decoder-Funktionen, die automatische Generierung eines Interface Control Document (ICD)[4], die Verwendung von ASN.1 Control Notation (ACN)[5], etc.[24, S.13ff]

Genauere Informationen zu ASN.1 und dem ASN1SCC können [24] entnommen werden.

5.2 Modellierung einer typischen Protokollkommunikation

Wie in Kapitel 4 beschrieben, beginnt die Entwicklung eines Systems mittels TASTE mit der Beschreibung der logischen Systemarchitektur und -schnittstellen. Dazu gehört die Modellierung des Datenmodells im Data View und die Definition der Systemfunktionen

[4] Ein ICD ist ein Dokument, dass die Daten, die zwischen kommunizierenden Softwarekomponenten ausgetauscht werden, beschreibt.[24, S.14]

[5] ACN ist eine Verschlüsselungsregel, die dem Entwickler erlaubt, das Format einer Nachricht auf Bit-Ebene zu kontrollieren.[24, S.14]

und deren Schnittstellen im Interface View.

Im Kapitel 5.2.1 wird die Erstellung des Datenmodells mittels ASN.1 für das Teilsystem des Beispielprojektes CHEOPS beschrieben.

Kapitel 5.2.2 umschreibt den Interface View des Beispielsystems. Dabei werden zuerst die Systemfunktionen mit ihren Eigenschaften dargestellt, anschließend werden die Schnittstellen mit ihren Eigenschaften beschrieben.

5.2.1 Erstellung des Datenmodells

Die Kommunikation der Komponenten eines Raumfahrtsystems erfolgt, wie bereits in Kapitel 3 beschrieben, über standardisierte Protokolle wie zum Beispiel das CCSDS/PUS-Protokoll. Dieses Protokoll erfüllt den Ende-zu-Ende Transport von Telekommando- und Telemetriepaketen zwischen Anwendungen am Boden und den Anwendungen der On-Board Software des Satelliten.[9, S.8] Die Telekommando- und Telemetriepakete müssen einer gewissen Struktur folgen, die im CCSDS 203.0–B–2[6] bzw. im CCSDS 102.0-B–5[7] Standard festgelegt und im Anhang A.1 und A.2 dargestellt sind.[9, S.24]

Die Kommunikationsprotokolle werden in einem Datenbanksystem des CHEOPS Projekts definiert, verwaltet und können zu Missionsdatenbanken der ESA exportiert werden. Aus diesen Datenbanken kann mittels eines Protokoll-Konverter-Scriptes eine benötigte ASN.1-Spezifikation einer Protokolldefinitionen exportiert werden. Genauere Informationen dazu können in [24] nachgelesen werden. Im Anhang A.3 befindet sich ein Ausschnitt der ASN.1-Spezifikation für ein Telekommando- und ein Telemetriepaket.

Diese ASN.1-/ACN-Spezifikation wurde für die Erstellung des Datenmodells des Beispielsystems genutzt. Dazu wurde mit folgendem Befehl in der Kommandozeile der Data View erzeugt: *taste-create-data-view*. In diesem wurde der Inhalt der ASN.1-Spezifikation kopiert. Mit dem Konsolenbefehl *taste-create-acn-model* wurde ein zugehöriges ACN-Modell erzeugt, in dem die ACN-Spezifikation kopiert wurde.

[6] http://public.ccsds.org/publications/archive/203x0b2s.pdf
[7] http://public.ccsds.org/publications/archive/102x0b5s.pdf

Während der Entwicklung des Beispielsystems wurde festgestellt, dass bei der eigenständigen Nutzung des ASN1SCC teilweise andere Konventionen gelten, als bei der Nutzung innerhalb der TASTE-Entwicklungswerkzeuge. Das muss bei der Verwendung der aus der Datenbank generierten ASN.1-/ACN-Spezifikationen beachtet werden.

So muss beispielsweise berücksichtigt werden, dass unter der Nutzung der TASTE-Werkzeuge alle Namen case-insensitive sind. Die Feldnamen in der ASN.1-Spezifikation sind somit nicht mehr ungleich den Typennamen. Als Lösung wurde zunächst händisch jedem Typnamen ein „T-“ als Kennzeichnung für einen Datentypen vorangestellt.

Der ASN.1 Typ NULL kann ebenfalls nicht im Datenmodell verwendet werden. Da dieser in der generierten ASN.1-Spezifikation genutzt wird, wurde ebenfalls zunächst händisch ein neuer Typ definiert. Dieser ist ein INTEGER-Typ, der nur den Wert „0“ zulässt.

Des weiteren muss beachtet werden, dass keine Enumerationen definiert werden, die die gleichen Werte enthalten. Es muss entschieden werden, ob man nur eine Enumeration für diese Werte definiert oder verschiedene Enumerationen, deren Werte angepasst werden müssen. Zur Lösung der Aufgabe wurde händisch jedem Enumerationswert der Name der Enumeration vorangestellt, sodass sich die Enumerationswerte verschiedener Enumerationen unterschieden.

Da die GUI laut einer Fehlernachricht bei der Generierung des Systems zum jetzigen Stand der Entwicklung noch keine Strings unterstützt, mussten alle String-Felder im Datenmodell entfernt werden, weil im Interface View des Beispielsystems eine GUI-Funktion definiert wurde.

Ein weiterer Punkt ist, dass Schlüsselwörter im Datenmodell nicht verwendet werden dürfen. Feldnamen oder Enumerationswerte wie z.B. „all“, „main“, etc. wurden zunächst händisch umbenannt. Welche Schlüsselwörter nicht erlaubt sind, wird in der Datei asnParser.py in der VirtualBox unter ∼/tool-src/DTM/commonPy festgelegt.[4, S.24]

Um die aus der Datenbank automatisch generierten ASN.1-/ACN-Spezifikationen weiterhin einsetzen zu können, muss das Script, das diese Spezifikationen erzeugt, diese

Konventionen beachten und dementsprechend angepasst werden.

In einer Konversation mit Herrn Sommer, einem Trainee der ESA, wurde erklärt, dass die VirtualMaschine auf die neuste TASTE-Version aktualisiert werden muss, um die *present-when*-Eigenschaft in der ACN-Spezifikation nutzen zu können. Dazu muss *Update-TASTE.sh* von der Kommandozeile aufgerufen werden.[25]

Trotz der anfänglichen Probleme mit den hinzugekommenen Konventionen des ASN1SCC konnte die vorhandene ASN.1-/ACN-Spezifikation mit einigen Modifizierungen für das Datenmodell verwendet werden.

Nachdem das Datenmodell mit den modifizierten ASN.1-/ACN-Spezifikationen erzeugt wurde, generierte TASTE aus diesem ein AADL-Modell für den Data View.[4, S.18]

5.2.2 Erzeugung des Interface Views

Für das Beispielsystem sollen die Services „Telecommand verification“, „Housekeeping and diagnostic data reporting“ und „Event reporting“ modelliert werden. Dazu wurden zunächst die Systemfunktionen erstellt und deren Eigenschaften wie z.B. die Ausführungssprache, Kontextparameter, etc. festgelegt. Insgesamt wurden für das Beispielsystem acht Systemfunktionen im Interface View erzeugt. Der Interface View des Beispielsystems ist im Anhang A.4 dargestellt.

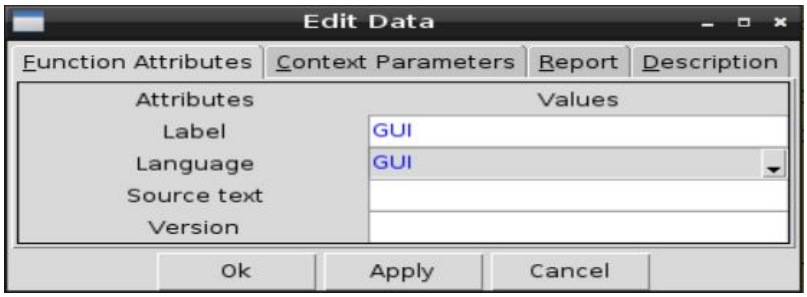


Abbildung 13: Eigenschaften einer Systemfunktion

Die Ausführungssprache einer Funktion, ihre Kontextparameter, etc. wurden im Interface View über die Eigenschaften der Systemfunktion definiert (Abbildung 13).

Im Schnittstellenmodell des Beispielsystems besitzt beispielsweise die Funktion „GUI“ die Ausführungssprache GUI. TASTE generiert in dem Fall automatisch eine GUI zum Versenden von Telekommandopakten und zum Empfangen von Telemetriepaketen. Für diese GUI wird bei der Erstellung des Systems eine eigene Binärdatei erzeugt, die beim Aufruf zusammen mit der System-Binärdatei mit dem laufenden System kommuniziert. Ein Bild der laufenden GUI ist in Abbildung 14 zu sehen.

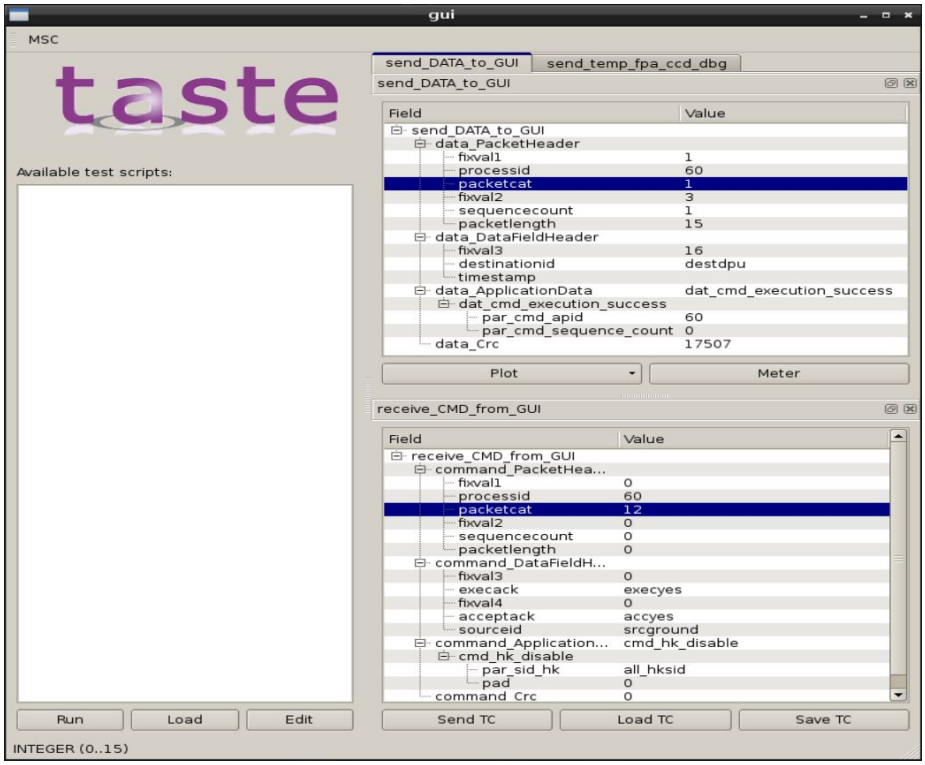


Abbildung 14: Automatisch generierte GUI zur Interaktion mit dem laufenden System

Des weiteren wurden die Funktionen, die keine Zustände annehmen und hauptsächlich Telemetriepakete packen, wie „DATAhandling“ „VerrificationService“ und „FDIR“ in C implementiert. Die restlichen Systemfunktionen wurden in der Ausführungssprache SDL implementiert, weil sie verschiedene Zustände annehmen können oder der Ablauf der Funktionen mit SDL übersichtlicher dargestellt werden kann.

Weil zur Umsetzung des „Housekeeping and diagnostic data reporting service“ zwei Periodenzeiten gesetzt werden können, besitzt die Funktion „HKservice“ zwei Kontext-

parameter vom Datentyp Timer. Diese Timer werden über bestimmte Telekommandos gesetzt. Nach Ablauf der Timer werden die entsprechenden Telemetriepakete über die „DATAhandling“-Funktion an die „GUI“-Funktion zurückgeschickt. Nähere Informationen zu dieser Funktion und zur Verwendung der Timer befinden sich in Kapitel 6.2.

Nachdem die Systemfunktionen erstellt und deren Eigenschaften definiert wurden, konnten die Schnittstellen der Funktionen spezifiziert werden.

Um zwei Systemfunktionen miteinander zu verbinden, muss eine Funktion ein RI und die andere Funktion ein PI besitzen. Diese Schnittstellen besitzen mehrere Eigenschaften, z. B.: Schnittstellenart, Minimum Inter-Arrival Time (MIAT)[8], die Deadline und die Worst Case Execution Time (WCET), Queuesize, etc.[4, S.22ff]

Es gibt vier verschiedene PI Schnittstellenarten [4, S.23]:

- Sporadic: asynchrone Schnittstelle; wird durch den Empfang von Daten aufgerufen
- Cyclic: asynchrone Schnittstelle; wird gemäß einer vordefinierten Periode aufgerufen
- Protected: synchrone Schnittstelle; wird exklusiv von einem Thread aufgerufen
- Unprotected: synchrone Schnittstelle; kann gleichzeitig durch verschiedene Threads aufgerufen werden

Für die Schnittstellenmodellierung des Beispielsystems wurden Sporadic und Cyclic Interfaces eingesetzt. Da für jedes Sporadic und jedes Cyclic Interface ein eigener Thread erzeugt wird, sind alle Schnittstellen innerhalb einer Funktion automatisch durch wechselseitigem Ausschluss geschützt.[4, S.23]

Im Beispielsystem wurden Cyclic Interfaces bei den Funktionen „HKservice“ und „CheopsTempFPAccdContol“ eingesetzt. Abbildung 15 zeigt die Einstellungen eines Cyclic Interfaces.

Die Periode eines Cyclic Interfaces gibt an, in welchen Zeitabständen die Schnittstelle aufgerufen wird. Die Deadline legt fest, wann die mit der Schnittstelle verbundene Aufga-

[8] Die MIAT gibt die minimale Zeit zwischen zwei Ereignissen an.[4, S.22]

be beendet werden muss. Die WCET ist die längst mögliche Ausführungszeit, die eine Schnittstelle besitzen kann.[4, S.69] Cyclic Interfaces besitzen keine Schnittstellenparameter.[4, S.23]

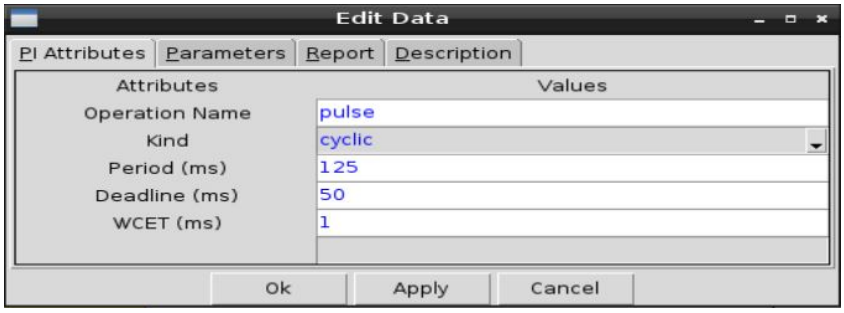
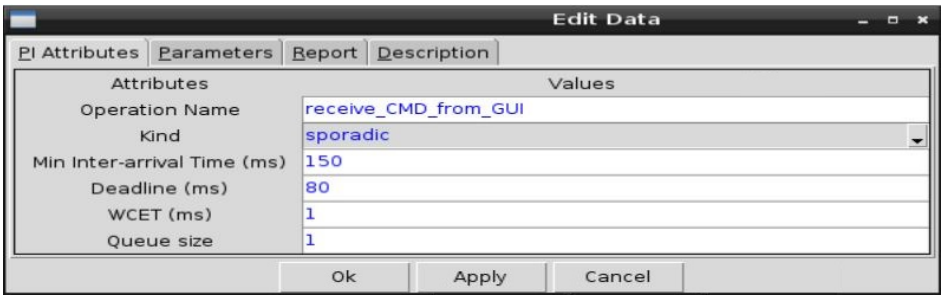


Abbildung 15: Editierung des Cyclic Interfaces

Im Gegensatz zum Cyclic Interface, verbindet ein Sporadic Interface zwei Funktionen miteinander. Abbildung 16 zeigt die Einstellungen eines PI und dem zugehörigen RI.



(a) Editierung des Provided Interfaces

(b) Editierung des Required Interfaces

Abbildung 16: Eigenschaften der Schnittstellen

Funktionen, die über die PI und die RI verbunden werden, können in unterschiedlichen Ausführungssprachen (z. B. C, Ada, VHSIC (Very High Speed Integrated Circuit)

Hardware Description Language (VHDL), Simulink, etc.) implementiert werden. Zur Übertragung der Daten zwischen den Systemfunktionen werden automatisch die Enocder- und Decoderfunktionen des ASN1SCC verwendet. Mittels dieser Funktionen können Systemfunktionen unterschiedlicher Ausführungssprachen problemlos Datenstrukturen austauschen.[4, S.21]

Welche Daten zwischen den Funktionen ausgetauscht werden, wird über die Parameter der Schnittstellen festgelegt. Dabei erbt das RI den Parameter des PI, sofern die Vererbung in den Eigenschaftes des RI zugelassen wird. Diese Parameter können nur Datentypen aus der ASN.1-Spezifikation besitzen.

In einer ersten Version des Systems wurden im Interface View zwischen den Funktionen „CMDhandling“ und „VerficationService“ ganze Telekommandopakte ausgetauscht. Für die weitere Verarbeitung der empfangenen Telekommandopakte in den folgenden Funktionen wurde nur der ApplicationData-Teil des empfangenen Paktes benötigt. Daher wurden lediglich benötigte Teile eines Paketes von einer Funktion zur nächsten weitergesendet. Zwischen der Funktion „DataHandling“ und der GUI-Funktion wurden komplette Telemetriepakete ausgetauscht. Für dieses System wurde im ACN-Modell die *present-when*-Eigenschaft der CHOICE-Typen weggelassen, da die Verwendung dieser Eigenschaft einen Fehler bei der Generierung des „Glue“-Codes erzeugte und somit das System nicht generiert werden konnte.

Wie bereits in Kapitel 5.2.1 erwähnt, konnte die *present-when*-Eigenschaft ohne Auftreten eines Fehlers genutzt werden, als die TASTE-Version auf die neuste Version aktualisiert wurde.[25] Die Generierung des Systems erfolgte jedoch nur problemlos, wenn zwischen den Funktionen immer ganze Pakete ausgetauscht wurden. Da dies im Realsystem ebenfalls der Fall ist, wurde das System mit der *present-when*-Eigenschaft im ACN-Modell und dem Austausch ganzer Pakete weiterentwickelt.

In der aktuellen Version des Beispielsystems (Anhang A.4) besitzen Schnittstellen, deren Namen mit „receive_CMD“ beginnen, Parameter vom Datentyp „T_Command_PACKET“ und werden somit zur Übertagung von Kommandopaketen verwendet. Datenpakete wer-

den von Schnittstellen übermittelt, die mit dem Namen „send_DAT“ beginnen. Sie besitzen einen Parameter vom Typ „T_Data_PACKET“. Schnittstellen, die Counter übertragen, besitzen einen Parameter vom Datentyp „T_unit16“. Vom Typ „T_Int32“ sind Schnittstellenparameter, die zu Übertragung der Temperaturwerte dienen. Schnittstellen wie „send_error“ und „receive_execution_error“ sind vorerst nur Platzhalter im Modell. Später werden über diese Schnittstellen Fehler, die in den Funktionen auftreten, signalisiert. Die Schnittstelle „send_temp_fpa_ccd_dbg“ dient zur Visualisierung der simulierten Temperaturwerte in der „CheopsTempFPAccdControl“-Funktion.

Neben dem Parameternamen und -typ, wurde für jeden Schnittstellenparameter das Verschlüsselungsprotokoll und die Richtung angegeben. Das Verschlüsselungsprotokoll jedes Schnittstellenparameters ist ACN. Die Richtung gibt an, ob der Parameter eines PI ein Eingangs- oder ein Ausgangsparameter einer Systemfunktion ist. Im Beispielsystem wurden nur Eingangsparameter verwendet, da Sporadic Interfaces keine Ausgangsparameter erlauben.[4, S.23]

Damit wurden für das Beispielsystem alle Datentypen und deren Einschränkungen im Datenmodell definiert und die Systemfunktionen mit allen Schnittstellen und Eigenschaften modelliert. TASTE generierte den Code für die Schnittstellen vollautomatisch. Außerdem wurden für jede definierte Funktion Codegerüste in der angegebenen Sprache erstellt. Diese Codegerüste mussten im nächsten Schritt vom Entwickler mit der eigentlichen Funktionalität befüllt werden.

6 Modellierung der Softwarefunktionalität

Für die Entwicklung eines Systems müssen die von TASTE automatisch generierten Codegerüste in der jeweiligen Sprache mit der gewünschten Funktionalität befüllt werden.

Bei der Implementierung der Funktionen soll die Modellierungssprache SDL genauer betrachtet werden. Einen Überblick über die SDL-Sprache bietet Kapitel 6.1

Welche Funktionalität für jede Funktion implementiert wurde und wie die Nutzbarkeit von SDL im Softwaremodell ausfällt, wird in Kapitel 6.2 beschrieben.

6.1 Überblick der Modellierungssprache SDL

Die Specification and Description Language (SDL) ist eine objektorientierte, formale Sprache, die vom ITU-T definiert wurde. Sie wird in Bereichen der Telekommunikation, der Satellitenkommunikation, in der Luftfahrt, der Medizin, in Zugkontrollsysteme und in Kommunikationsprotokollen für Autos genutzt.[26, S.1,S.4]

SDL besitzt sowohl eine grafische als auch eine textuelle Notation und dient zur formalen Beschreibung der Struktur, des Verhaltens und der Daten eines echtzeitfähigen, verteilten Kommunikationssystems. SDL als formale Sprache sichert Präzision, Konsistenz und Klarheit im Design, das wesentlich für missionskritische Systeme ist.[26, S.4ff]

Systementwickler definieren das System als Zustandsmaschine. Diese besitzen einfache Konzepte, feste Grundlagen, verstecken Implementationsdetails und konzentrieren sich auf die Dynamik eines Systems.[27, S.2] Ein SDL-Diagramm bietet somit dem Entwickler einen guten Überblick über den Ablauf eines Systems.

Das theoretische SDL-Grundmodell besteht aus Zustandsmaschinen, die parallel laufen und unabhängig voneinander sind. Die Kommunikation dieser Zustandsmaschinen erfolgt über diskrete Signale.[26, S.5]

Im allgemeinen Gebrauch beschreibt SDL die Struktur eines Systems mittels vier Strukturlevel: System, Block, Prozess, Prozedur. Eine statische Struktur wird mit Systemen, Blöcken und Kanälen beschrieben, bei der Blöcke als Black-Boxen aufgefasst werden.

Eine dynamische Struktur, wie sie im TASTE-Prozess modelliert werden kann, wird mit Prozessen und Signalen definiert.[26, S.6ff] In Abbildung 17 ist ein Beispiel eines SDL-Diagramms eines Prozesses zu sehen.

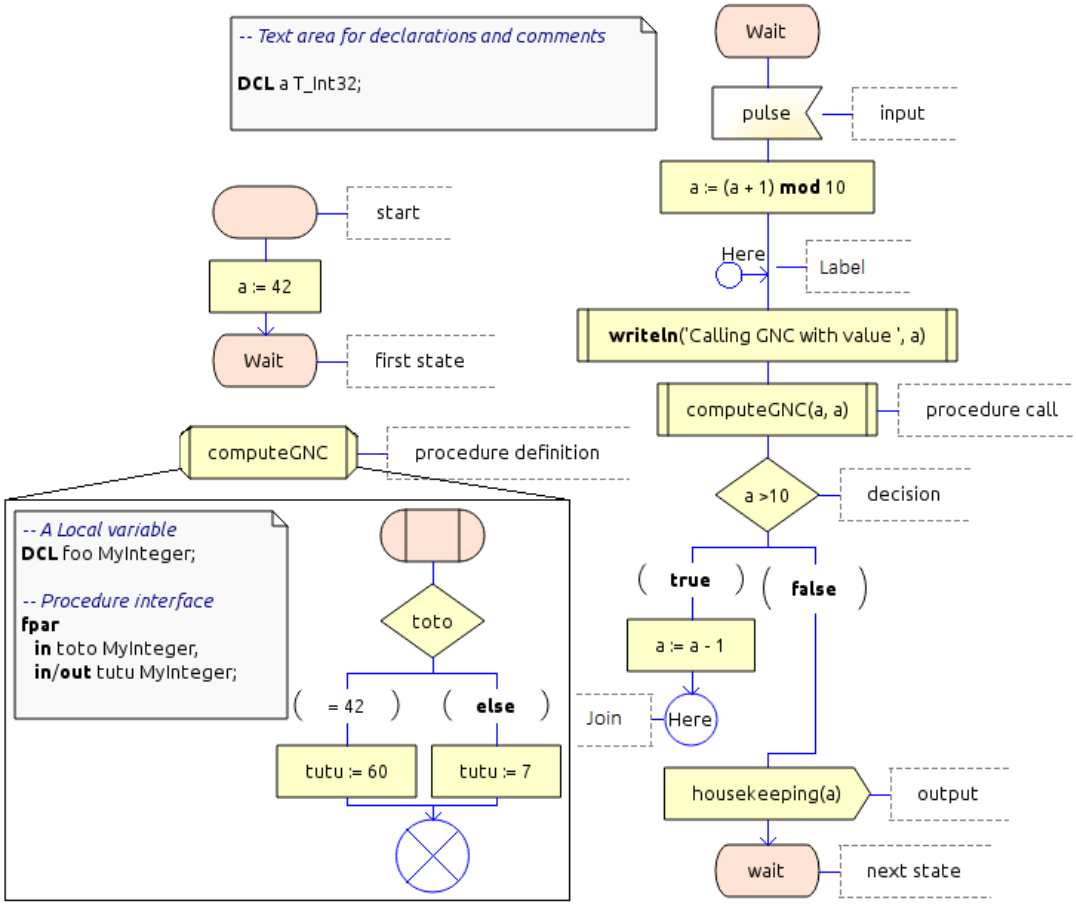


Abbildung 17: Beispiel eines SDL-Modells

Mit den Kanälen und Signalen, die optionale Parameter besitzen können, wird die Kommunikation eines Systems modelliert. SDL benutzt keine globalen Variablen. Die Kommunikation erfolgt über asynchrone Signale oder entfernte Prozeduraufrufe.[26, S.7]

Zur Beschreibung der Daten werden abstrakte Datentypen wie Abstract Data Type (ADT)[1] oder ASN.1 verwendet. ASN.1 erlaubt einen Datenaustausch zwischen verschiedenen Sprachen und bietet die Möglichkeit Datenstrukturen wiederzuverwenden.[26, S.9]

[1] Abstract Data Type ist ein Datentyp mit keiner spezifizierten Datenstruktur.[26, S.9]

In TASTE werden die Daten mit ASN.1 beschrieben.

OpenGEODE ist ein grafischer Editor, in dem ein System mit SDL modelliert und beschrieben werden kann. Er ist frei, open-source, leicht zu handhaben [27, S.9] und wird von TASTE als integrierter SDL-Editor für einfache Funktionen angeboten.[28, S.18] OpenGEODE besitzt einen Semantik- und Syntaxprüfer, der Nichtdeterminismus, toten Code und die Gefahr von Speicherüberlauf statisch entdeckt.[27, S.19]

6.2 Erstellung eines Beispiel-Softwaremodells mit SDL

Im Rahmen der Aufgabenstellung soll die Nutzung von SDL anhand eines Softwaremodells für das Beispielprojekt CHEOPS getestet werden.

Zur Lösung der Aufgabe wurden Funktionen im Interface View (Anhang A.4) definiert, die beispielhaft den „Telecommand verification service“, den „Housekeeping and diagnostic data reporting service“ und den „Event reporting service“ umsetzen und im folgenden näher beschrieben werden.

„GUI“-Funktion

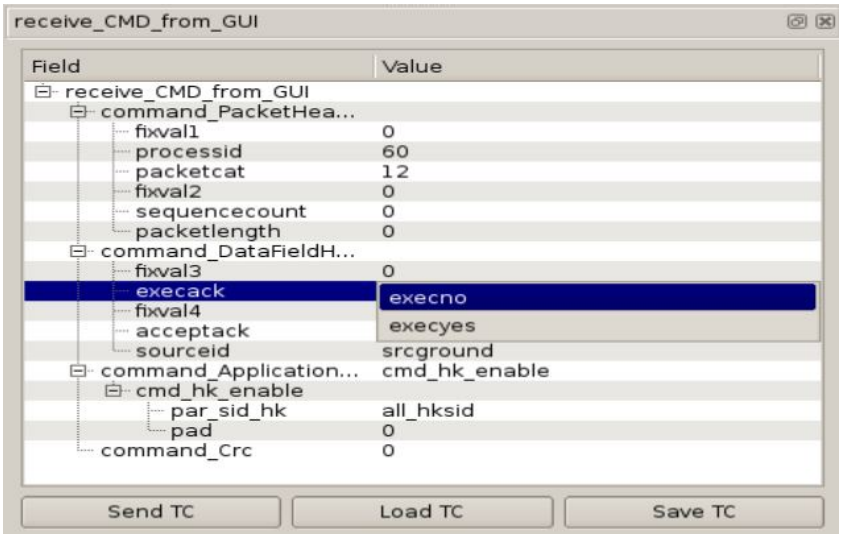


Abbildung 18: Ausschnitt der GUI zum Senden der Telekommandopakete

Die „GUI„-Funktion besitzt die Ausführungssprache GUI. TASTE generiert automatisch für diese Funktion eine vollständige GUI zum Senden und Empfangen von Paketen.[4, S.11] Von dieser GUI werden Telekommandopakete an die Funktion „CMDhandling“ gesendet.

Daten, die im Paket stehen sollen, werden in der GUI eingetragen. Dabei können nur ASN.1 Datentypen verwendet werden. D. h., dass bereits in der GUI nur zulässige Werte eingetragen oder ausgewählt werden können.

Abbildung 18 zeigt einen Ausschnitt der Oberfläche zum Senden von Telekommandopaketen. Dabei ist zu sehen, dass beispielsweise für Enumerationen nur Werte ausgewählt werden können, die im Datenmodell definiert wurden.

Über die Schnittstelle „send_DATA_to_GUI“ empfängt die Funktion Telemetriepakete von der Funktion „DATAhandling“. Der Inhalt der empfangenen Pakete wird wie beim Senden in der GUI dargestellt.

„CMDhandling“-Funktion

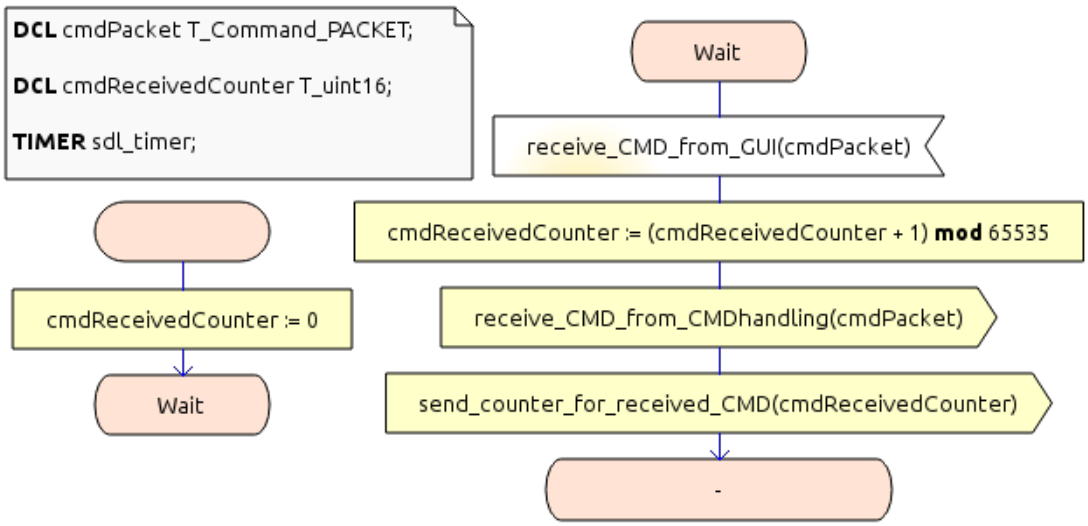


Abbildung 19: SDL-Diagramm der „CMDhandling“-Funktion

Die „CMDhandling“-Funktion wurde, wie in Abbildung 19 zu sehen ist, in SDL implementiert. Sie empfängt und zählt die gesendeten Telekommandopakete der „GUI“-Funktion.

Bei der Initialisierung der Funktion wird der Zähler auf null gesetzt und in den „Wait“-Zustand gewechselt. Wird in diesem Zustand ein Kommandopaket über die Schnittstelle „receive_CMD_from_GUI“ empfangen, wird der Zähler um eins erhöht und Modulo 65535 gerechnet, um den Wertebereich des Zählertyps nicht zu überschreiten. Anschließend wird das empfangene Paket über die „receive_CMD_from_CMDhandling“-Schnittstelle an die „VerificationService“-Funktion gesendet. Der Zähler wird zum Schluss an die „HKservice“-Funktion gesendet und die „CMDhandling“-Funktion bleibt im Zustand „Wait“.

Diese Funktion soll zu einem späteren Zeitpunkt die empfangenen Telekommandopakete buffern, bis die nachfolgenden Funktionen bereit sind diese zu verarbeiten. Sollte dabei ein Fehler auftreten, weil beispielsweise der Buffer voll ist, soll ein Fehler an die „FDIR“-Funktion gesendet werden. Für das Beispielsystem ist dieser Buffer nicht notwendig, da die beispielhafte Bearbeitung der Kommandos nicht viel Zeit in Anspruch nimmt und sofort neue Kommandos verarbeitet werden können.

„VerificationService“-Funktion

Die „VerifikationService“-Funktion implementiert den „Telecommand verification service“ des „ECSS-E-70-41A“-Standards (Siehe [9]). Dabei wird nur der minimale Servicesatz verwendet. D. h., dass die Funktion einzig entscheidet, ob ein Telekommandopaket angenommen wird. Die „VerifikationService“-Funktion wurde in C implementiert. Der Quellcode dieser Funktion befindet sich im Anhang A.5.

Wie bereits in Kapitel 3 beschrieben müssen bei diesem Service verschiedene Parameter des empfangenen Paketes auf ihre Richtigkeit überprüft werden. Dies wird mit verschiedenen If-Anweisungen realisiert. Ist mindestens ein fehlerhafter Parameter aufgetreten, wird ein „DAT_CMD_Acceptance_Failure“-Paket, das einen entsprechenden Fehlercode enthält, mit allen paketspezifischen Parametern gepackt und an die „DATAhandling“-Funktion geschickt. Ein „DAT_CMD_Acceptance_Success“-Paket wird mit allen paketspezifischen Parametern an die „DAThandling“-Funktion gesendet, wenn alle Parameterwerte korrekt sind und das Feld „acceptack“ im DataFieldHeader den Wert „accyes“ enthält, also ein

Akzeptanzbericht erwartet wird. Dies wird mit einer weiteren If-Abweisung überprüft. Nur im Falle einer erfolgreichen Paketannahme wird das empfangene Telekommandopaket an die Funktion „CMDdispatcher“ weitergesendet.

„CMDdispatcher“-Funktion

Die „CMDdispatcher“-Funktion wertet die empfangenen Telekommandos aus und entscheidet vorrangig, an welche Funktion die Pakete weitergeleitet werden sollen. Diese Funktion wurde aus gründen der Übersichtlichkeit in SDL implementiert. Das zugehörige SDL-Diagramm befindet sich in A.6.

Wie im SDL-Diagramm zu sehen ist, wird mittels einer SDL-Entscheidung geprüft, welchen ApplicationData-Typ das Paket besitzt. Je nachdem welcher Typ vorhanden ist, wird das Paket an eine entsprechende Funktion weitergeleitet. Besitzt ein empfangenes Paket beispielsweise den Typ „cmd_hk_disable“, wird das Paket über die Schnittstelle „receive_CMD_HK_Disable“ an die Funktion „HKservice“ gesendet.

Nach dem Weitersenden eines Paketes wird ein Zähler erhöht und Modulo 65535 gerechnet, um den Wertebereich des Datentyps nicht zu überschreiten. Der Zähler zählt alle Pakete, die erfolgreich erkannt und weitergeschickt werden konnten. Dieser Zähler wird an die „HKservice“-Funktion gesendet.

Anschließend wird mit einer SDL-Entscheidung geprüft, ob ein Ausführungsbericht angefordert wird. Dazu wird der Wert im Feld „execack“ des DataFieldHeaders des empfangenen Paketes abgefragt. Ist dieser Wert „execyes“, wird ein „DAT_CMD_Execution_Success“-Paket an die „DATAhandling“-Funktion gesendet.

Wird bei der Abfrage der ApplicationData des empfangenen Paketes ein unbekannten Typ festgestellt, wird ein „DAT_CMD_Execution_Failure“-Paket an die Funktion „DATAhandling“ gesendet. Da im aktuellen Modell kein spezieller Ausführungsfehlercode, z. B. *inconsistent* oder *not executable*, genutzt werden kann, weil diese Fehler im Beispielsystem nicht auftreten können, besitzt das Paket nur default Werte.

„HKservice“-Funktion

Der „Housekeeping and diagnostic data reporting service“ des „ECSS-E-70-41A“- Standards (Siehe [9]) wird von der HKservice-Funktion realisiert. Die Funktionen dieses Services sind in Kapitel 3 beschrieben. Die HKservice-Funktion ist ebenfalls in SDL implementiert. Das SDL-Diagramm dieser Funktion befindet sich in Anhang A.7.

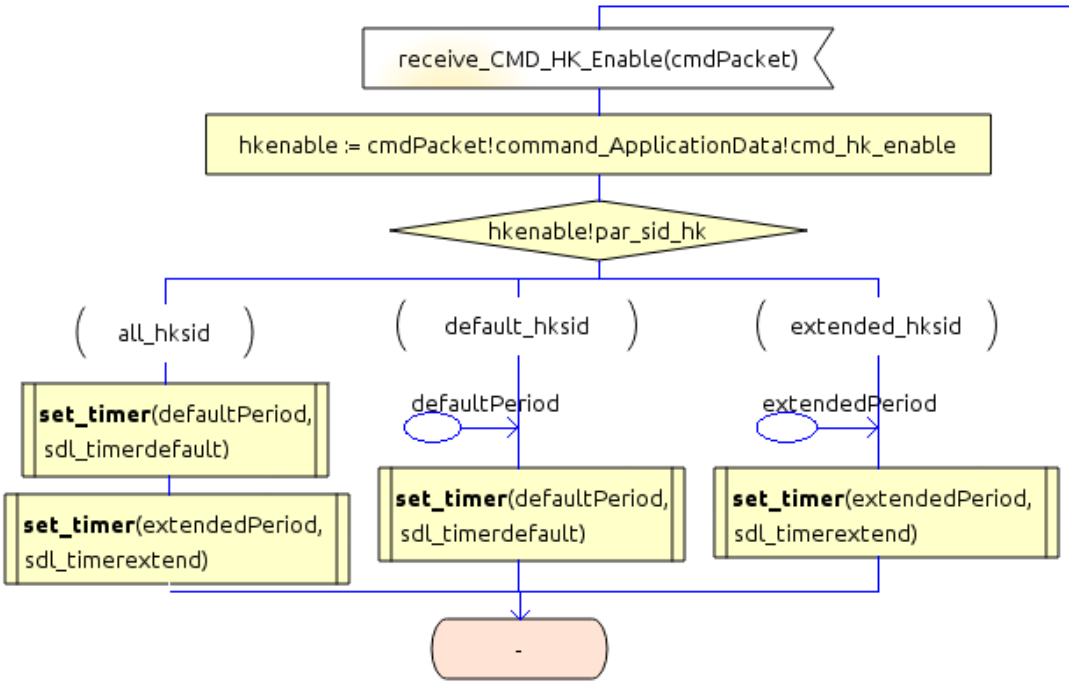


Abbildung 20: SDL-Diagramm zur Verarbeitung eines „CMD_HK_Enable“-Paketes

Die „HKservice“-Funktion besitzt zwei Timer. Ein Timer ist für die Periode der Default-Housekeepingdaten (defaultHK), der andere für die Extended-Housekeepingdaten (extendedHK). Wird ein „CMD_HK_Enable“-Paket empfangen, wird je nach Angabe des „par_sid_hk“-Parameters in den ApplicationData, einer der beiden Timer oder beide gesetzt. Die Überprüfung der Parameter erfolgt mittels einer SDL-Entscheidung. Zum Setzen der Timer, wird die gebotene *set_timer*-Funktion genutzt. Dieser Teil der HKservice-Funktion wird von Abbildung 20 dargestellt.

Nach Ablauf der Timer erhält der SDL-Prozess ein Signal, wie er jedes andere Eingangssignal erhält. Demnach wird der Timer wie ein Eingangssignal gehandelt.[26, S.7]

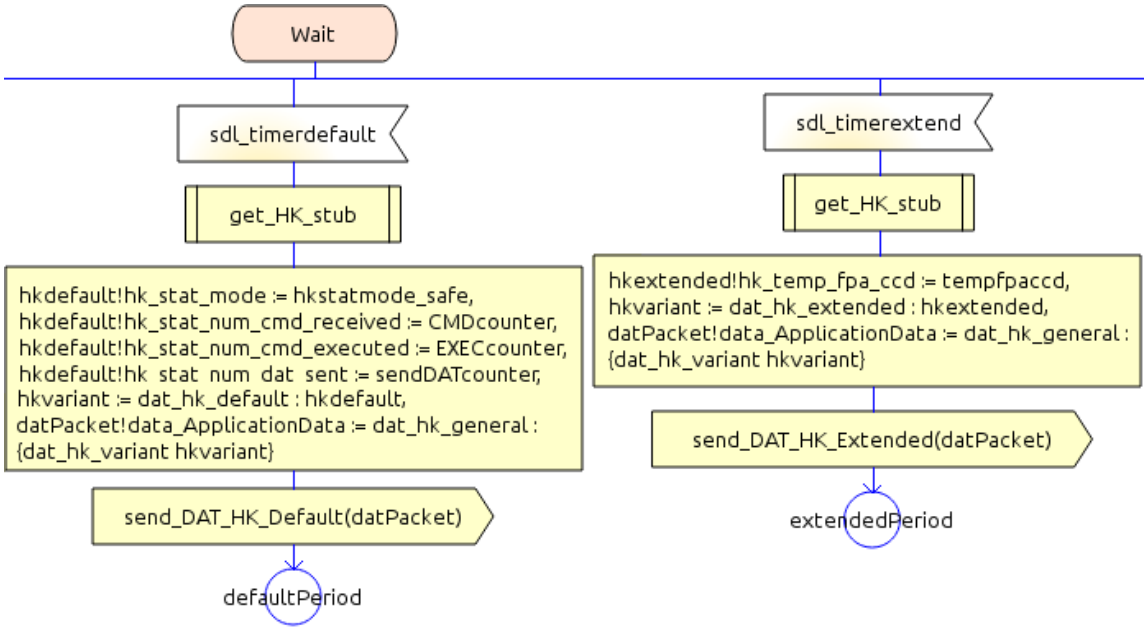


Abbildung 21: SDL-Diagramm zur Verarbeitung der empfangenen Timer-Signale

Empfängt die „HKservice“-Funktion ein Timer-Signal, wird eine Prozedur aufgerufen, die die Housekeepingdaten generiert. Im Beispielsystem ist diese Prozedur nur ein Platzhalter, da keine realen Messwerte aufgenommen werden. In einem realen System muss dieser Platzhalter durch eine Funktion zur Telemetriedatenerfassung ersetzt werden. Nach dem Aufruf der Prozedur werden, je nachdem welches Timer-Signal empfangen wurde, die paketspezifischen Parameter eines Paketes für die defaultHK oder eines Paketes für die extendedHK definiert und über die entsprechende Schnittstelle an die Funktion „DATAhandling“ gesendet. Mit einer *Join*-Anweisung wird zum zugehörigen *Label* gesprungen und ab dort die folgenden Anweisungen abgearbeitet. Das bedeutet, dass nach dem Senden der Pakete zu der Stelle gesprungen wird, an der der abgelaufene Timer erneut gesetzt wird. In Abbildung 21 sind die eben beschriebenen Anweisungen dargestellt. In Abbildung 20 sind die zu den *Join*-Anweisungen zugehörigen *Label* zu sehen, an denen nach dem Senden der Pakete gesprungen wird.

Für jeden Timer wurde eine Variable erzeugt, die angibt nach wie vielen ms ein Timer abläuft. Bei der Initialisierung der Funktion werden beide Variablen auf 20000 gesetzt. Mit

dem Empfang eines „CMD_HK_Period“-Paketes wird dieser Wert verändert. Zunächst wird durch die Abfrage des Parameter „par_sid_hk“ aus den ApplicationData des empfangenen Paketes mit einer SDL-Entscheidung bestimmt, welcher Timerwert geändert werden soll. Der Wert des Parameters „par_hk_period“ aus dem ApplicationData des Paketes wird eine der beiden Variablen oder beiden zugewiesen. Abbildung 22 zeigt den eben beschriebenen Teil der Funktion.

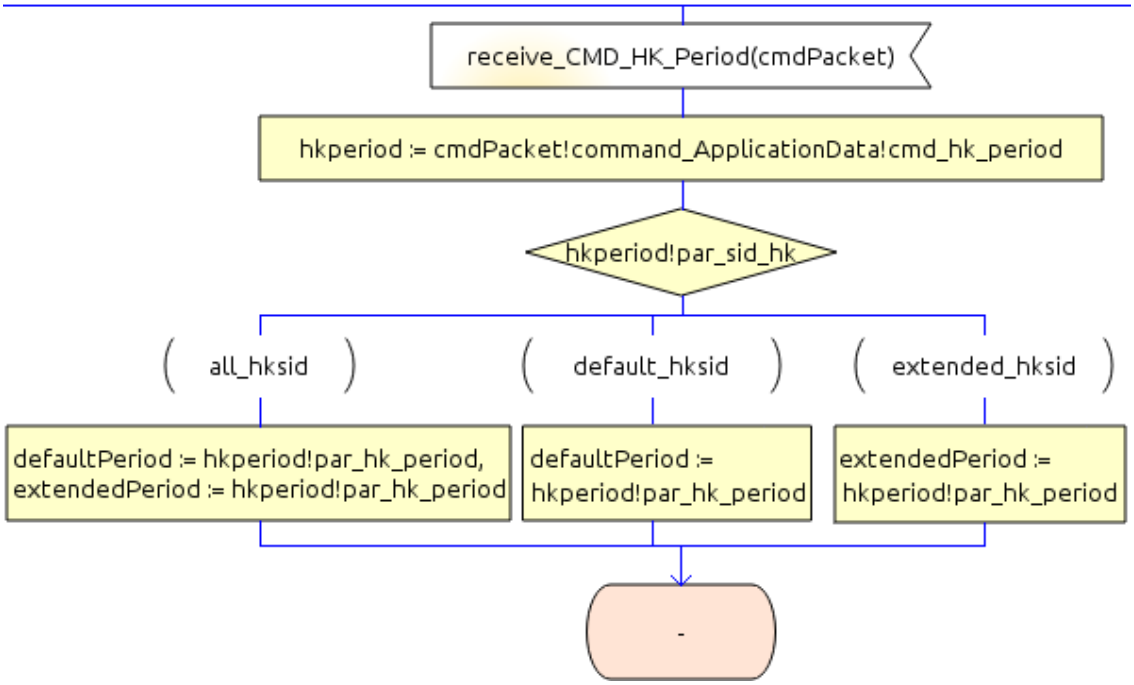


Abbildung 22: SDL-Diagramm zur Verarbeitung eines „CMD_HK_Period“-Paketes

Bei der Angabe der Periode muss beachtet werden, dass der Timer-Wert ein Vielfaches von 100 ms sein muss. Die 100 ms sind das vom Zeitmanager genutzte Zeitquantum. Der Zeitmanager ist eine Funktion mit einer periodischen Schnittstelle, die alle 100 ms ausgeführt wird. Diese Funktion hält alle Timer-Zustände und ist dafür verantwortlich, nach Ablauf eines Timers die Schnittstelle im Benutzercode aufzurufen.[29]

Um die Datenaufnahme zu stoppen wird ein „CMD_HK_Disable“-Paket gesendet. Mittels einer SDL-Entscheidung wird der Wert des Parameters „par_sid_hk“ des empfangenen Paketes geprüft. Es wird entschieden, welche Datenaufnahme (defaultHK und/oder

extendedHK) gestoppt werden. Dazu wird der entsprechende Timer mittels der gebotenen *reset_timer*-Funktion zurückgesetzt wie es Abbildung 23 zeigt.

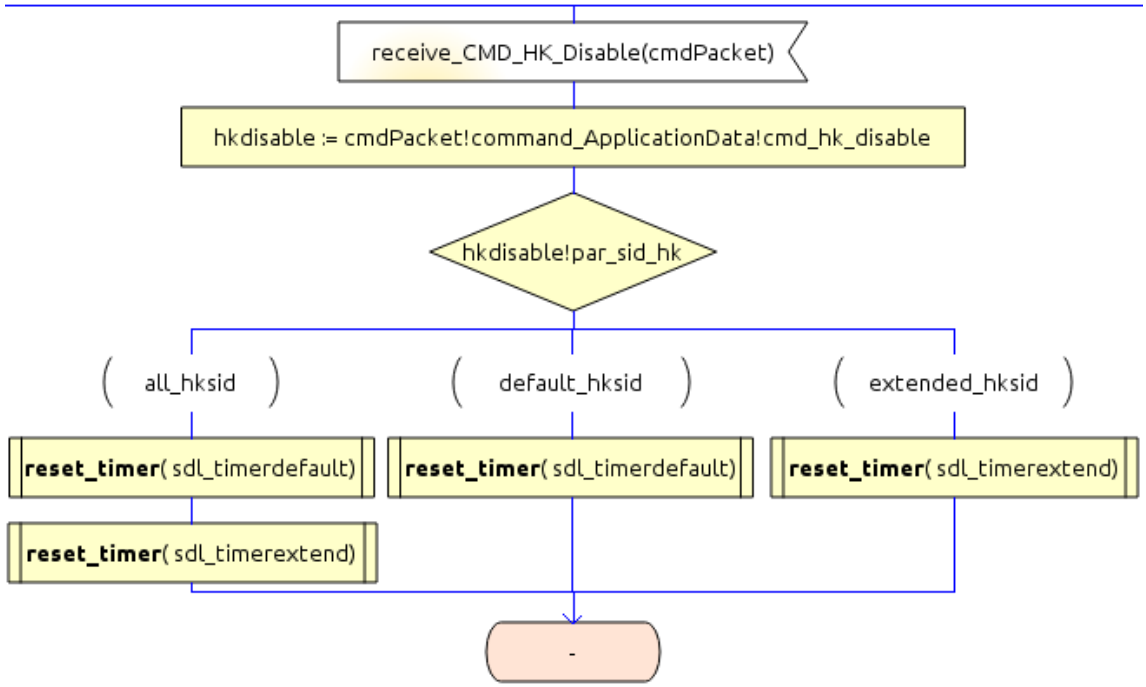


Abbildung 23: SDL-Diagramm zur Verarbeitung eines „CMD_HK_Disable“-Paketes

Die „HKservice“-Funktion besitzt des weiteren Schnittstellen zum Empfang der Zähler der „CMDhandling“- und der „CMDdispatcher“-Funktion sowie zum Empfang der Temperaturwerte aus der „CheopsTempFPAccdControl“-Funktion. Beim Eingang eines Zählers bzw. eines Temperaturwertes wird der empfangene Wert einer zugehörigen Zähler- bzw. Temperatur-Variablen zugeteilt, deren Wert einem entsprechenden Parameter in einem zu sendenden „DAT_CMD_HK_Default“-Paket zugewiesen wird.

Weiterhin besitzt die Funktion ein Cyclic Interface, dass alle 125 ms aufgerufen wird. Diese Schnittstelle ist dazu gedacht, alle 125 ms Housekeepingdaten aufzunehmen. Im Beispielsystem wird sie zunächst nur aufgerufen und besitzt keine weitere Funktionalität. Im real System muss auch hier eine Funktion zur Telemetriedatenerfassung eingefügt werden.

„CheopsTempFPAccdControl“-Funktion

Die „CheopsTempFPAccdControl“-Funktion (Anhang A.8) ist in SDL implementiert und modelliert die Regelung einer Temperaturkontrolle.

Dabei gibt es zwei Zustände:

- „TempControlDisabled“, wenn der gemessene Temperaturwert nicht kontrolliert wird
- „TempControlEnabled“, wenn der gemessene Temperaturwert kontrolliert wird

Die Funktion besitzt zwei Cyclic Interfaces. Wird die „pulse_calc_temp“-Schnittstelle in einen der beiden Zustände aufgerufen, wird die „fget_temp_fpa_ccd_stub“ Prozedur (Abbildung 24) aufgerufen.

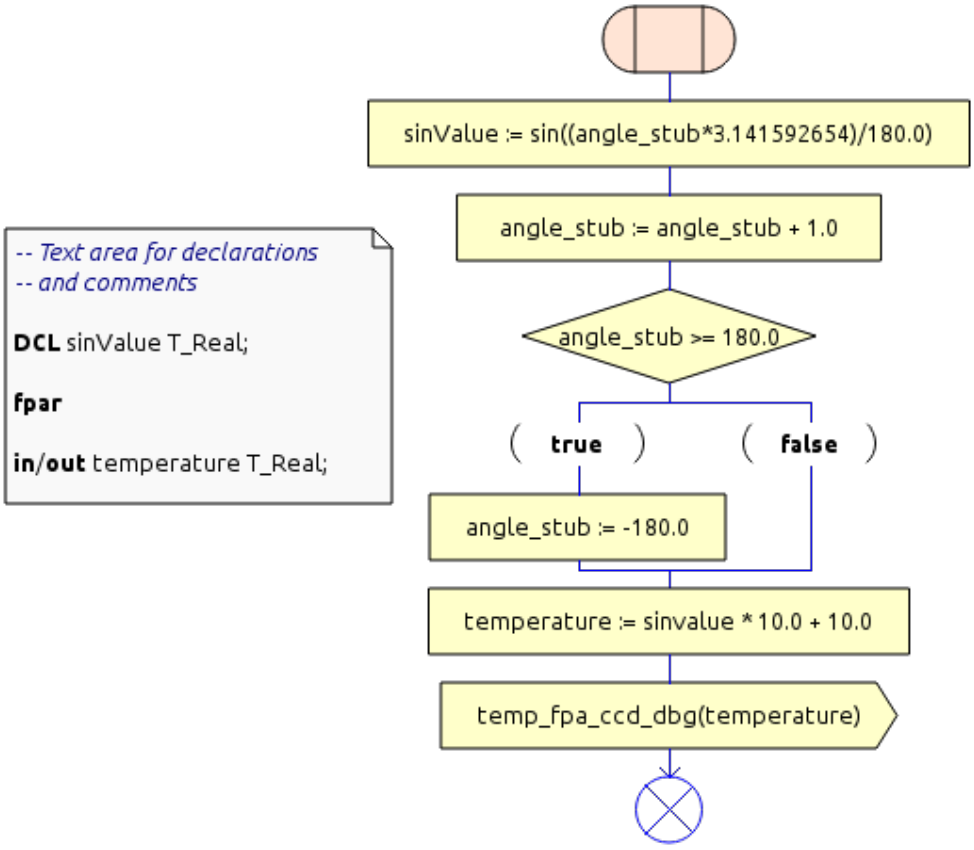


Abbildung 24: SDL-Diagramm der Prozedur „fget_temp_fpa_ccd_stub“

Diese Prozedur berechnet im Beispielsystem Sinus-Werte und gibt diese zurück an den

aufrufenden Prozess. An dieser Stelle könnten beispielsweise Messdaten des realen Systems aus einer Datei gelesen werden, um den Regelalgorithmus anhand von realistischen Daten zu testen. In einem realen System muss dieser Stub durch eine Funktion zur Temperaturmessung ersetzt werden.

Die „CheopsTempFPAccdControl“-Funktion prüft nach dem Aufruf der Prozedur mit einer SDL-Entscheidung, ob der simulierte Temperaturwert größer gleich 20 ist. In diesem Fall wird der Temperaturwert an die Funktion „FDIR“ gesendet, die dann ein „DAT_Event_Warning“-Paket mit entsprechendem Warnungshinweis packt.

Befindet sich die Systemfunktion im „TempControlDisabled“-Zustand und erhält ein „CMD_Temp_Control_Enable“-Paket, so wechselt die Funktion in den Zustand „TempControlEnabled“.

Wird im Zustand „TempControlEnabled“ das Cyclic Interface „pulse“ aufgerufen, wird mit einer SDL-Entscheidung geprüft, welchen Wert die von der „fget_temp_fpa_ccd_stub“-Prozedur zurückgelieferte Temperaturvariable besitzt. Nach dem Verhalten eines Zweipunktschalters mit Hysterese wird entschieden, welche Anweisungen folgen. Ist der Temperaturwert kleiner als 10, wird im Beispielsystem die Ausgabe „Heater on!“ durchgeführt. An dieser Stelle müsste ein Heizelement eingestellt werden. Ist der Temperaturwert größer als 15, müsste ein Heizelement ausgestellt werden. Schließlich wird der Temperaturwert an die „HKservice“-Funktion gesendet.

Wird ein „CMD_Temp_Control_Disable“-Paket empfangen, wechselt die Funktion in den Zustand „TempControlDisabled“ zurück.

„FDIR“-Funktion

Die „FDIR“-Funktion des Systems realisiert den „Event reporting service“ des „ECSS-E-70-41A“-Standards (Siehe [9]). Diese Funktion sendet ein „DAT_Event_Warning“-Paket an die „DATAhandling“-Funktion, wenn die „send_temperature_fpa_ccd_err“-Schnittstelle aufgerufen wird. In diesem Fall ist der Temperaturwert aus der „CheopsTempFPAccdCon-

trol“-Funktion zu hoch und es muss eine entsprechende Warnung gesendet werden.

Die „FDIR“-Funktion ist dafür verantwortlich alle paketspezifischen Parameter in das „DAT_Event_Warning“-Paket zu schreiben und es über die „send_DAT_Event_Warning“-Schnittstelle an die „DATAhandling“-Funktion zu senden.

Im Beispielsystem sendet diese Funktion Fehler als Datenpakete an die BEE. Dies ist in C einfacher zu implementieren als mit SDL. Der zugehörige Quellcode befindet sich in Anhang A.9.

Die restlichen Schnittstellen der Funktion sind im Beispielsystem vorerst ohne weitere Funktionalität, da das Beispielsystem nur einen Teil der APS modelliert und nicht allen Schnittstellen eine Aufgabe zugewiesen werden kann.

„DATAhandling“-Funktion

Die „DATAhandling“-Funktion (Quellcode im Anhang A.10) ist eine in C implementierte Funktion, die alle Datenpakete, die zur „GUI“-Funktion gesendet werden sollen, an diese schickt. Wird eines der PI aufgerufen, vervollständigt die Funktion das eingegangene Paket und sendet es über die „send_DAT_to_GUI“-Schnittstelle zur „GUI“-Funktion.

Die „DATAhandling“-Funktion kann beispielsweise dazu genutzt werden die Checksumme jedes Datenpaketes zu berechnen und diese an das Paket anzufügen.

Über die Schnittstelle „send_counter_for_send_DATA“ wird ein Zähler an die Funktion „HKservice“ geschickt. Dieser wird für jedes an die „GUI“-Funktion gesendete Paket um eins erhöht und Modulo 65535 gerechnet, um den Wertebereich einzuhalten.

Dieses Beispielsystem zeigt, dass sich die Nutzung von SDL zum Erstellen von Softwaremodellen mit dem Gebrauch des grafischen Editors OpenGEODE aus den TASTE-Entwicklungswerkzeugen als aussichtsreich erweist, obwohl sich dieser noch in der Entwicklungsphase befindet.

Die intuitive, grafische Notation von SDL, wie sie von TASTE genutzt wird, bietet einen

guten Überblick über den Ablauf und das Verhalten einer Funktion und ist an einigen Stellen weniger aufwendig als beispielsweise eine Implementierung in C. Mit einem SDL-Entscheidungssymbol kann der Wert einer Variablen ohne großen Aufwand übersichtlich überprüft werden. Durch die Nutzung der ASN.1 Typen werden beispielsweise Missachtungen des Wertebereichs sofort als Fehler gekennzeichnet. Bei der Nutzung des OpenGEODE-Editors ist zu beachten, dass „ - “ in Feld- und Typnamen in der ASN.1-Spezifikation in „ _ “ umgewandelt werden.

Es ist möglich alle Funktionen des Beispielsystems in SDL zu implementieren. Jedoch wäre die Programmierung einiger Funktionen etwas aufwendiger, da in einer SDL-Funktion lediglich die in der ASN.1-Spezifikation definierten Datentypen verwendet werden können. In einer C-Funktion hingegen kann beispielsweise auf die automatisch generierten Funktionen zum en- und decoden der ASN.1 Typen zugegriffen werden, was Datenzuweisungen in C einfacher und übersichtlicher macht als in einer SDL-Funktion. Aus diesem Grund wurden Funktionen, die lediglich Datenzuweisungen durchführen, in C implementiert.

Weiterhin zeigt dieses Beispielsystem, dass Funktionen, die in verschiedenen Sprachen implementiert wurden mittels der TASTE-Werkzeuge ohne großen Aufwand des Entwicklers miteinander kommunizieren können.

7 Simulation und Test des Softwaremodells

Zum Testen des entstandenen Softwaremodells muss zunächst die Ausführungsumgebung festgelegt und die Systemfunktionen den zu verwendenden Prozessoren zugewiesen werden. Um das System zunächst lokal zu testen, wurde im Deployment View ein x86 Prozessor mit Linux als Betriebssystem gewählt und alle Funktionen des Interface Views hinzugefügt. Der resultierende Deployment View ist in Abbildung 25 zu sehen.

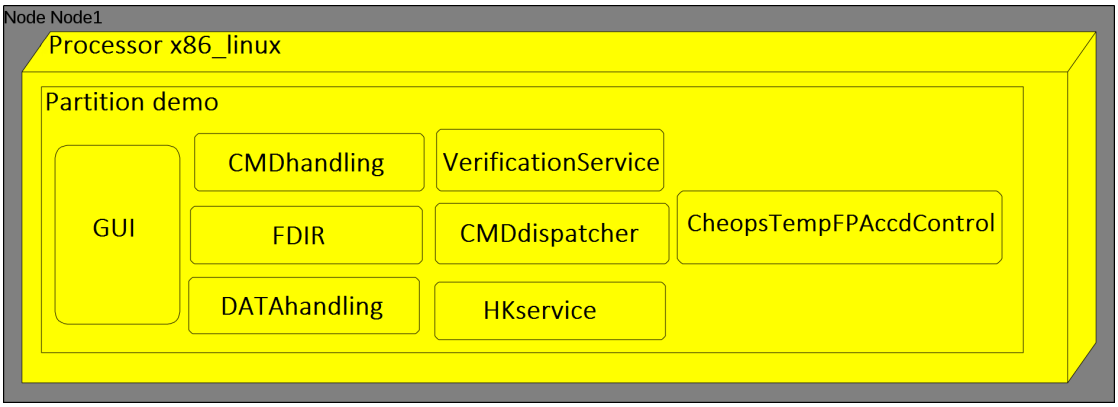


Abbildung 25: Deployment View des Beispielsystems

Nachdem die Systemfunktionen mit ihren Eigenschaften und Schnittstellen definiert und den Hardwarekomponenten zugewiesen wurden, erstellt TASTE aus der abstrakten Anwendungsbeschreibung (dem Interface View und dem Deployment View) ein Modell (Concurency View), das die speziellen Ausführungskomponenten (Tasks, gemeinsame Variablen, etc.) enthält. Das AADL-Modell des Concurrency View kann mit Cheddar hinsichtlich der Echtzeitfähigkeit des Systems analysiert werden.[14, S.5ff] Das Cheddar Analysewerkzeug und die Schedulability-Analyse des Beispielsystems werden in Kapitel 7.1 beschrieben.

Nach einer erfolgreichen Schedulability-Analyse kann das System in C oder Ada erzeugt und getestet werden. In Kapitel 7.2 werden die TASTE-Werkzeuge zum Testen des erzeugten Programmcodes vorgestellt.

7.1 Schedulability-Analyse mit Cheddar

Wie bereits in Kapitel 4.1 erwähnt, ist Cheddar ein Schedulability-Analysewerkzeug, das vom ConcurrenyView-Editor angeboten wird. Cheddar ist ein freies, in Ada geschriebenes Echtzeit-Scheduling-Werkzeug. Cheddar beruht auf Ocarina (Kapitel 4.1), um die Schedulability-Analyse von AADL-Modellen bereitzustellen.[30] Es simuliert die zeitliche Planung von Tasks und prüft das zeitliche Verhalten von Echtzeitanwendungen, die oft zeitliche Einschränkungen wie Antwortzeit, Ausführungsraten oder Deadlines einhalten müssen. Das Cheddar-Werkzeug wurde von einem Team der Universität Brest und Ellidiss Technologies entwickelt.[31, S.4]

In Cheddar repräsentiert jeder periodische Task t_i eine Parallelоperation mit einer definierten Deadline d_{t_i}, einer Periode P_{t_i} und einer Ausführungszeit C_{t_i}. Der Task t_i wird alle P_{t_i} Zeiteinheiten geweckt und führt seine Aufgabe, deren Ausführung durch C_{t_i} Zeiteinheiten begrenzt wird, aus. Die Aufgabe muss D_{t_i} Zeiteinheiten nach dem Erwecken erledigt sein.[31, S.4]

Cheddar besitzt zwei Fähigkeiten. Zum einen die Ausführung von Durchführbarkeitstests zur Analyse der Echtzeitanwendung ohne die zeitliche Planung eines Tasks zu berechnen. Durchführbarkeitstests können anstelle von Scheduling-Simulationen angewandt werden, wenn beispielsweise eine Scheduling-Simulation zu lang zum Berechnen ist. Die Cheddar-Durchführungstests basieren auf verschiedene Eigenschaften wie der Auslastungsfaktor eines Prozessors, die Antwortzeit eines Tasks, der Auslastungsfaktor des Zwischenspeichers, etc.[31, S.4]

Zum anderen bietet Cheddar eine Simulationsengine zum Vorhersagen, welcher Task, für jede Zeiteinheit und jeden Prozessor, ausgeführt wird und auf welchem Prozessor er zur Ausführung kommt. Cheddar kann die zeitliche Planung mit dafür üblichen Echtzeitalgorithmen simulieren. Dabei werden einige nützliche Informationen wie die schlechteste, beste, durchschnittliche Antwortzeit, blockierte Zeit, Anzahl der Kontextwechsel, Auslastungsfaktor des Zwischenspeichers, etc. zur Verfügung gestellt.[31, S.4]

Für jedes Ergebnis der Schedulability-Analyse gibt Cheddar den Namen der verwendeten Gleichung zur Schedulability-Simulation und zugehörigen Publikation an, in der die Gleichung zu finden ist. Funktionalitäten der Scheduling-Simulation mit klassischen Echtzeit-Scheduler, die auch in TASTE verwendet werden, sind beispielsweise Rate Monotonic Analysis (RMA)[1] und Deadline Monotonic (DM)[2]. [30] Weitere Informationen zu den verwendbaren Algorithmen sind unter http://beru.univ-brest.fr/~singhoff/cheddar/ zu finden.

Zur Analyse des Concurrency Views führt Cheddar Durchführbarkeitstests durch, indem die Task Beschreibungen mit modernen Validierungsalgorithmen, beispielsweise RMA, verarbeitet werden. Damit wird ein Auslastungsfaktor des Prozessors berechnet und die Durchsetzung von zeitliche Voraussetzungen (z. B. Task-Deadline) bewertet.[14, S.5]

Die Cheddar-Analyse für das Beispielsystem ergab einen Prozessorauslastung-Faktor von 0.69498 bei der Verwendung des RMA (Abbildung 26).

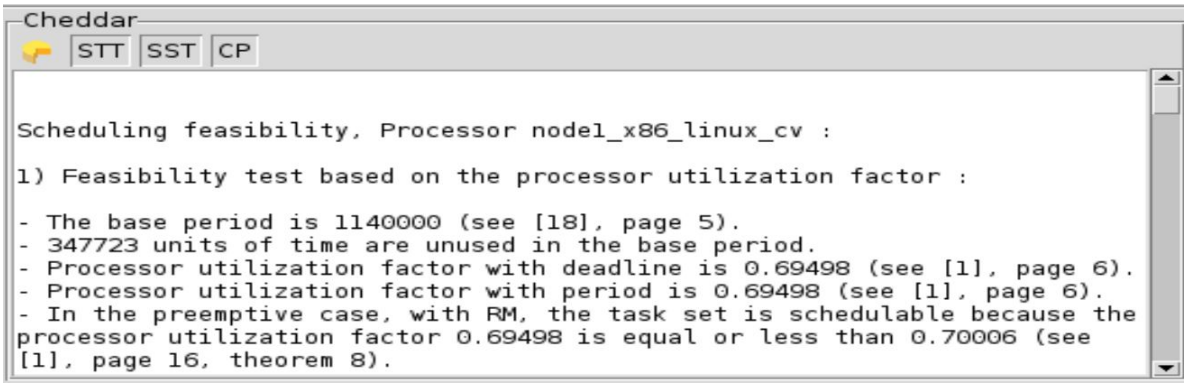


Abbildung 26: Ausschnitt aus der Scheduling-Analyse mit Cheddar

Der Auslastungsfaktor U wird unter der Verwendung des RMA wie folgt berechnet:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (\sqrt[n]{2} - 1) \tag{7.1}$$

[1] RMA ist auf eine Methode zurückzuführen, die Prioritäten einer Reihe von Prozessen als eine monotonische Funktion der Rate eines Prozesses zuweist.[32, S.3]

[2] DM ist der RMA ähnlich. Prioritäten werden Prozessen invers proportional der Deadline-Länge zugewiesen.[33, S.2]

Mit C_i als Ausführungszeit eines Tasks, T_i als Periode eines Tasks und n die Anzahl der Tasks. [32, S.5]

Die Schedulability-Analyse mit Cheddar bietet den Vorteil, eine Prüfung der Echtzeitfähigkeit eines Systems vorzunehmen, ohne mit dem realen System zu arbeiten und beispielsweise eine Überbelastung zu simulieren.

7.2 Testen des erzeugten Programmcodes

Orchestrator automatisiert den Build-Prozess eines Systems und erzeugt eine ausführbare Binärdatei. Beim Aufruf des Orchestrator-Werkzeugs können diesem verschiedene Parameter übergeben werden. Zu diesen Parametern gehören „–with-coverage“ und „–gprof“.[4, S.85]

Mit dem Parameter „–with-coverage“ werden die entsprechenden Coverageoptionen des GCC genutzt, um den Aufruf von gcov auf die erzeugte Binärdatei zu erlauben.[4, S.86] gcov ist ein Test Coverage Programm des GCC und ermöglicht Code-Coverage für C-Code.[24, S.29]

Mit dem Parameter „–gprof“ werden Binärdateien erzeugt, die mit gprof profiliert werden können. gprof erlaubt die Durchführung einer Performanceanalyse des Systems.[4, S.86]

Bei der Umsetzung der Aufgabe wurde festgestellt, dass diese Funktionen von TASTE nicht unterstützt werden. In einer E-Mail hat die ESA (Herr Perrotin) diese Vermutung bestätigt. Zugleich wurde darauf hingewiesen, dass die Entwickler auch mit Werkzeugen, die nicht zu der TASTE-Werkzeugsammlung gehören, arbeiten und eine Integration der Rapitime-Werkzeuge (Kapitel 4.2) veröffentlichen wollen. Diese ermöglichen, viele Metriken (z.B. WCET, Überdeckung, etc.) von der Ausführung des Codes zu erstellen.[34]

8 Anwendung der TASTE Entwicklerwerkzeuge für das CHEOPS-Projekt

Nachdem ein Teil der CHEOPS APS mit den TASTE-Werkzeugen entwickelt wurde, wird in diesem Kapitel anhand der bisherigen Ergebnisse ein möglicher Prozess zur Entwicklung der vollständigen CHEOPS APS beschrieben. Dabei wird hauptsächlich auf zu beachtende Problemstellungen eingegangen.

Bei der Entwicklung der CHEOPS APS ist der sogenannte TASTE-Prozess zu beachten. Dieser umfasst alle wichtigen Systementwicklungsschritte, die bei der Nutzung von TASTE zur Softwareentwicklung befolgt werden müssen. Genauere Informationen zum TASTE-Prozess sind in Kapitel 4 gegeben.

Zur CHEOPS APS Entwicklung bietet TASTE die TASTEGUI, die den Benutzer durch den Softwareentwicklungsprozess leitet.

Nach Rücksprache mit der ESA (Herrn Perrotin einem TASTE-Entwickler) wurde eingeräumt, dass diese Funktionalität in den TASTE-Werkzeugen noch nicht vollständig entwickelt ist.[34] Alternativ kann jedoch die CHEOPS APS Entwicklung mit spezifischen Konsolenbefehlen durchgeführt werden.

Zur Entwicklung der CHEOPS APS wird zuerst mit dem Konsolenbefehl *taste-create-dataview* ein Texteditor zur Erstellung des CHEOPS Datenmodell erstellt. Für die Definition der verwendeten Datentypen im Datenmodell soll die aus der CHEOPS Datenbank automatisch generiere ASN.1-/ACN-Spezifikation verwendet werden. Das gewährleistet die Konsistenz im Entwicklungsprozess der CHEOPS APS. Nachdem die ASN.1-Spezifikation des CHEOPS Projektes in das Datenmodell kopiert wurde, kann mit dem *taste-create-acn-model*-Konsolenbefehl ein Texteditor zur Erstellung des ACN-Modells geöffnet werden. Die ACN-Spezifikation des CHEOPS Projektes wird in das ACN-Modell kopiert.

Alternativ können die Namen der ASN.1-Datei und der ACN-Datei, die die ASN.1- und

ACN-Spezifikationen des CHEOPS Projektes enthalten, in „DataView.asn“ bzw. „DataView.acn“ umbenannt werden. In diesem Fall müssen in der Konsole lediglich in dem Verzeichnis, in dem sich die Dateien befinden, die eben genannten Befehle aufgerufen werden. Im geöffneten Texteditor sind dann die CHEOPS ASN.1- bzw. die CHEOPS ACN-Spezifikation bereits vorhanden. Im selben Verzeichnis wird nach dem Schließen des Texteditors das AADL-Modell für den CHEOPS Data View erzeugt.

Für eine Nutzung der automatisch generierten CHEOPS ASN.1-/ACN-Spezifikation im Datenmodell, muss das Script, das diese Spezifikation automatisch aus der CHEOPS Datenbank generiert, aufgrund der erweiterten Konventionen zur Nutzung des ASN1SCC innerhalb der TASTE-Werkzeuge, angepasst werden. Diese modifizierten Konventionen und Anpassungsvorschläge sind in Kapitel 5.2.1 benannt.

Anschließend wird mit dem Konsolenbefehl *taste-create-interface-view* der InterfaceView-Editor geöffnet. In diesem Editor werden die CHEOPS Systemfunktionen und deren Eigenschaften modelliert. Welche Funktionseigenschaften definiert werden können und wie die CHEOPS Systemfunktionen miteinander verbunden werden, ist in Kapitel 5.2.2 nachzulesen.

Der Interface View Editor stellt eine Funktion bereit, um das modellierte CHEOPS System in eine Bilddatei zu konvertieren. Diese kann zu Dokumentationszwecken des CHEOPS Softwaredesigns genutzt werden.

Um eine CHEOPS Systemfunktion mit der entsprechenden Funktionalität zu befüllen, kann über einen Rechtsklick auf die jeweilige Funktion ein weiteres Werkzeug für die entsprechende Implementierungssprache geöffnet werden. D. h., bei einer CHEOPS Systemfunktion mit der Ausführungssprache C kann über einen Rechtsklick auf die Funktion der Texteditor „Kate“ geöffnet werden, um den C-Code der Systemfunktion zu bearbeiten. Der OpenGEODE-Editor kann durch den Rechtsklick auf eine CHEOPS Systemfunktion mit der Sprache SDL geöffnet werden. In Kapitel 6.2 ist beschrieben, was bei der Nutzung des OpenGEODE-Editors zu beachten ist. Zum Beispiel können in einer SDL-Funktion nur Datentypen aus der CHEOPS ASN.1-Spezifikation genutzt werden.

Wie Beispielmodellierung im Kapitel 6.2 zeigt, können verschiedene Temperaturregelalgorithmen für die CHEOPS Software simuliert und ausprobiert werden, ohne dass das CHEOPS System gebaut wurde.

Zur Festlegung der Ausführungsumgebung für das CHEOPS System kann über das Menü des InterfaceView-Editors oder über den Konsolenbefehl *taste-create-deployment-view* im entsprechenden Projektverzeichnis der DeploymenView-Editor geöffnet werden. Dieser bietet eine Auswahl an verschiedenen Prozessoren, Treibern und Bussen. Die CHEOPS Funktionen können problemlos verschiedenen Prozessoren zugewiesen werden. Diese Prozessoren müssen über Treiber und Busse miteinander verbunden werden. Der Austausch der Hardware kann ohne großen Aufwand durchgeführt werden. Beispielsweise können die CHEOPS Funktionen, die auf einem x86-Prozessor ausgeführt wurden, problemlos einem Leon-Prozessor zugewiesen werden.

Auch der DeploymentView-Editor bietet eine Funktion, um den CHEOPS Deployment View in eine Bilddatei zu exportieren, die dann zu Dokumentationszwecken genutzt werden kann.

Über das Menü des DeploymentView-Editors oder über den Konsolenbefehl *taste-edit-concurrency-view* kann der sogenannte ConcurrencyView-Editor geöffnet werden. Dieser bietet wie in Kapitel 4.1 vorgestellt verschiedene Werkzeuge zur Verifizierung des CHEOPS Systems. Mit dem TASTE-Werkzeug Cheddar kann eine Schedulability-Analyse zur Überprüfung des Echtzeitverhaltens des CHEOPS Systems durchgeführt werden. Welche Funktionen Cheddar bietet kann in Kapitel 7.1 nachgelesen werden.

Wie bereits in Kapitel 7.2 beschrieben ist zurzeit eine Code- und Performanceanalyse des entwickelten CHEOPS Systems mit den TASTE-Werkzeugen nicht möglich. Alternativ kann das lizenzpflichtige Rapitime-Werkzeug (Kapitel 4.2) zur statischen und dynamischen Analyse der CHEOPS APS verwendet werden.

Es ist möglich eine ausführbare Binärdatei einer GUI-Funktion des CHEOPS Interface Views erstellen zu lassen. Diese kann zusammen mit der CHEOPS Systembinärdatei

aufgerufen werden, um den Austausch der Telekommando- und Telemetriepakete mit der CHEOPS APS zur Laufzeit zu simulieren. Wie in Kapitel 4 beschrieben, können die empfangenen Telemetriedaten über die GUI mittels GnuPlot zur Laufzeit überwacht und der Nachrichtenaustausch durch den MSC Tracer kontrolliert werden.

Schließlich kann die erzeuge Systemdatei der CHEOPS APS auf die Zielplattform portiert werden.

9 Fazit

Dieses Kapitel bietet eine Zusammenfassung der Ergebnisse der Aufgaben dieser Arbeit und gibt einen Ausblick über weitere Aspekte, die zur Nutzung der TASTE-Werkzeuge betrachtet werden können.

9.1 Zusammenfassung

Nach einer umfassenden Einarbeitung konnte mit den TASTE-Entwicklungswerkzeugen ein Teilsystem des Beispielprojektes CHEOPS erfolgreich modelliert werden. Das Beispielsystem realisiert eine Schnittstellenkommunikation nach ESA-Standards, als einen Teil der APS On-Board Software.

Für die Entwicklung des Beispielsystems wurde zunächst das Datenmodell erstellt. Beim Import der vorhandenen ASN.1-/ACN-Spezifikation in das Datenmodell des Beispielsystems wurde festgestellt, dass bei der Nutzung des ASN1SCC innerhalb der TASTE-Entwicklungswerkzeuge teils andere Konventionen gelten, als bei der selbstständigen Nutzung des ASN1SCC. Die erforderlichen Änderungen wurden aus Zeitgründen zunächst händisch angepasst und müssen im Protokoll-Konverter-Script geändert werden. Nach den Änderungen im Script kann die automatisch generierte ASN.1-/ACN-Spezifikation problemlos für das Datenmodell des CHEOPS Projektes genutzt und somit die Konsistenz der Protokolldefinition im Entwicklungsprozess gewährleistet werden.

Für die Modellierung der Schnittstellenkommunikation zwischen den Systemfunktionen, musste zunächst die TASTE-Version auf die Version vom Mai 2014 aktualisiert werden, um die standardisierte Protokollkommunikation des Beispielsystems problemlos erstellen zu können.

Die definierten Systemfunktionen des Beispielsystems modellieren beispielhaft einige Services des ESA-Standards ECSS-E-70-41A, wie sie auch im CHEOPS Projekt genutzt werden. Das Kommando- und Datenhandling des Beispielprojektes konnte relativ unkompliziert modelliert werden. Des weiteren ist es möglich mit diesem Modell z. B. einen

Temperaturregelungsalgorithmus in einer Art und Weise auszuprobieren, wie man sie in der On-Board Software nutzt, ohne das reale, sehr aufwendige und i. d. R. in wenigen Stückzahlen verfügbare Hardwaresystem nutzen zu müssen.

Die Modellierung der Systemfunktionen hat weiterhin gezeigt, dass die Modellierungssprache SDL mit der formalen Syntax und Semantik einen guten Überblick über das Verhalten einer Funktion liefert und so andere Entwickler des Systems auch ohne Kenntnis der SDL-Sprache schnell erkennen, wie diese Funktionalität umgesetzt ist. SDL eignet sich gut zum schnellen und einfachen modellieren von Funktionen mit verschieden Zuständen.

Um eine Verifikation der entwickelten Beispielsoftware durchzuführen, wurde zunächst definiert, welcher Hardware die Systemfunktionen zugewiesen werden. Es werden verschiedene Hardwarekomponenten von den TASTE-Werkzeugen angeboten. Die gewählten Hardwarekomponenten können ohne großen Aufwand in kurzer Zeit ausgetauscht werden. Somit können verschiedene Hardwarekomponenten für das entwickelte System getestet werden, ohne dass die Hardware real zur Verfügung stehen muss. Die jeweiligen Hardwareaspekte werden bei der Simulation und dem Test des Beispielsystems von den TASTE-Werkzeugen beachtet.

Das entwickelte Teilsystem des CHEOPS Projektes wurde mit dem Cheddar-Werkzeug auf die zeitliche Ausführbarkeit analysiert. Cheddar beachtet die Eigenschaften der gewählten Hardware bei der Analyse. Dadurch können die Zeitberechnungen und Analysen simuliert werden, ohne das reale System gebaut zu haben. Eine frühzeitige Verifikation des Systems ist somit möglich.

Die Testfunktionen der TASTE-Werkzeuge sind zum jetzigen Zeitpunkt noch nicht funktionsfähig. Jedoch wurde von den Entwicklern darauf hingewiesen, dass das Werkzeug Rapitime zur statischen und dynamischen Analyse des Codes in die TASTE-Werkzeuge integriert werden soll.

Die TASTE-Werkzeuge können demnach für die Modellierung der CHEOPS Software genutzt werden, um effizient und ohne großen Entwicklungsaufwand die Anforderungen der ESA-Standards zu erfüllen. Die TASTE-Werkzeuge übernehmen viele Aufgaben der Entwickler, beispielsweise die Erzeugung von Encoder- und Decoder-Funktionen zur Kommunikation der unterschiedlichen Systemteile, wodurch der Entwicklungsaufwand sinkt. Durch das Angebot verschiedener Analyse-Werkzeuge, können bereits in einer frühen Entwicklungsphase Verifikationen des Systems erfolgen. Dadurch werden einige Nachweise zur Einhaltung der Qualitätsanforderungen, wie z. B. der Nachweis zur Einhaltung der definierten Antwortzeit des Systems, vereinfacht und beschleunigt. Es ist jedoch darauf hinzuweisen, dass TASTE immer noch in der Entwicklungsphase ist und nach Aussage der Entwickler noch nicht zur Entwicklung von produktiver Software für reale Systeme eingesetzt werden sollte. Viele Funktionen funktionieren noch nicht nachvollziehbar zuverlässig und einige sind noch nicht vollständig implementiert. Es ist nicht ausgeschlossen einige Funktionen, beispielsweise den Interface View, zu Dokumentations- oder Anschauungszwecken des CHEOPS Softwaredesigns zu verwenden.

Die Untersuchung der TASTE-Werkzeuge hat gezeigt, dass die Entwicklung der TASTE-Werkzeuge vielversprechend erscheint und das modellbasierte Softwareentwicklungsmethoden in Zukunft bei fehlerfrei funktionierenden Entwicklungsumgebungen bzw. Werkzeugen ein unverzichtbarer Bestandteil der Softwareentwicklung sein wird. Das trifft insbesondere zu, wenn Systeme mit hoher technischer Kritikalität und Qualitätsanforderung oder von mittleren und großen Teams entwickelt werden müssen.

9.2 Ausblick

Der Einsatz von TASTE zur Entwicklung einer Raumfahrtsoftware, wie sie in OS entwickelt werden, hängt von einigen Faktoren wie Codesize, Performance, etc. des erzeugten Softwarecodes und vor allem auch vom Entwicklungs- und Verifikationsstand von TASTE selbst ab. Diese Faktoren wurden bei der Auswertung der Ergebnisse nicht berücksichtigt.

Bei weiteren Analysen, beispielsweise einer Performanceanalyse, sollen diese beachtet und ausgewertet werden.

Da aus Zeitgründen keine Portierung auf die reale Hardwareplattform machbar war, ist dies ein weiterer Punkt, der getestet werden muss, um das Ziel, aus einem Modell eine lauffähigen Software zu erhalten, letztendlich bestätigen zu können. Dazu müssen die Systemfunktionen des Beispielsystem im Deployment View einem Leon-Prozessor zugewiesen werden. Anschließend muss die ausführbare Binärdatei für das System neu erzeugt werden, die dann auf die reale Hardwareplattform portiert und dort getestet wird.

Wie bereits in Kapitel 5.2.1 erwähnt, muss das Protokoll-Konverter-Script an die veränderten Konventionen zur Nutzung des ASN1SCC angepasst werden, damit die Konsistenz der Protokolldefinition zur Kommunikation im Entwicklungsprozess gewährt werden kann.

Außerdem kann die Nutzbarkeit weiterer TASTE-Werkzeuge wie der MSC Tracer von PragmaDev zur Echtzeitüberwachung des Nachrichtenaustauschs oder Marzhin zur Simulation der zeitlichen Systemplanung, etc. zur Entwicklung einer Software getestet werden.

Neben den internen TASTE-Werkzeugen können die nicht zur TASTE-Werkzeugsammlung gehörenden Werkzeuge wie Rapitime zur Verifikation, etc. zur Softwareentwicklung mit TASTE getestet und bewertet werden.

Des weiteren sollten die TASTE-Entwicklungswerkzeuge anhand der Entwicklung eines weiteren Beispielprojektes bewertet werden, um zu prüfen, ob ähnliche Projekte ebenfalls mit den TASTE-Werkzeugen erfolgreich entwickelt werden können.

Die weitere Entwicklung der TASTE-Werkzeuge verspricht ein Werkzeug zur einer effizienten Entwicklung von Raumfahrtsoftware, die den Qualitätsanforderungen der ESA gerecht wird, wenn TASTE selbst den Qualitätsanforderungen künftig entsprechen wird. Ein zukünftiger Einsatz der Werkzeugsammlung zur Entwicklung von OS-Raumfahrtprojekten wäre dann nach weiteren Evaluierungsschritten ein annehmbarer Weg zur Entwicklung einer modellgetriebenen Software.

A Anhang

A.1 Telekommandopaketstruktur

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

CCSDS Secondary Header Flag	TC Packet PUS Version Number	Ack	Service Type	Service Subtype	Source ID	Spare
Boolean (1 bit)	Enumerated (3 bits)	Enumerated(4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Enumerated (n bits)	Fixed BitString (n bits)
					← Optional →	← Optional →

Abbildung 27: Telekommandopaketstruktur nach dem CCSDS 203.0–B–2 Standard [9, S.42-44]

A.2 Telemetriepaketstruktur

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Source Data	Spare (Optional)	Packet Error Control (Optional)
Version Number (=0)	Type (=0)	Data Field Header Flag	Applica-tion Process ID	Grouping Flags	Source Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	(see Note 2)

Spare	TM Source Packet PUS Version Number	Spare	Service Type	Service Subtype	Packet Sub-counter	Destination ID	Time	Spare
Fixed BitString (1 bit)	Enumerated (3 bits)	Fixed BitString (4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Unsigned Integer (8 bits)	Enumerated	Absolute Time	Fixed BitString (n bits)
					Optional	Optional	Optional	Optional

Abbildung 28: Telemetriepaketstruktur nach dem CCSDS 102.0-B–5 Standard [9, S.46-48]

A.3 Ausschnitt aus der CHEOPS ASN.1-Spezifikation

```
1  TASTE-Dataview DEFINITIONS AUTOMATIC TAGS::= BEGIN
2  T-uint4 ::= INTEGER(0..15)
3  T-uint5 ::= INTEGER(0..31)
4  T-uint7 ::= INTEGER(0..127)
5  T-uint8-t ::= INTEGER(0..255)
6  T-uint14 ::= INTEGER(0..16383)
7  T-uint16 ::= INTEGER(0..65535)
8  T-timestamp ::= OCTET STRING (SIZE(6))
9  T-NULL ::= INTEGER(0)
10
11 -- Command-Paket
12 T-Command-PACKET ::= SEQUENCE {
13   command-PacketHeader  T-Command-PacketHeader,
14   command-DataFieldHeader T-Command-DataFieldHeader,
15   command-ApplicationData T-Command-ApplicationData,
16   command-Crc T-Crc }
17
18 -- Data-Paket
19 T-Data-PACKET ::= SEQUENCE {
20   data-PacketHeader T-Data-PacketHeader,
21   data-DataFieldHeader  T-Data-DataFieldHeader,
22   data-ApplicationData  T-Data-ApplicationData,
23   data-Crc  T-Crc }
24
25 -- Command-PacketHeader
26 T-Command-PacketHeader ::= SEQUENCE {
27   fixval1 T-uint5,
28   processid T-uint7,
29   packetcat T-uint4,
30   fixval2 INTEGER(0..3),
31   sequencecount T-uint14,
32   packetlength  T-uint16 }
```

```
33
34 -- Data-PacketHeader
35 T-Data-PacketHeader ::= SEQUENCE {
36   fixval1 T-uint5,
37   processid T-uint7,
38   packetcat T-uint4,
39   fixval2 INTEGER(0..3),
40   sequencecount T-uint14,
41   packetlength  T-uint16 }
42
43 -- Command-DataFieldHeader
44 T-Command-DataFieldHeader ::= SEQUENCE {
45   fixval3 T-uint4,
46   execack T-Command-ExecAck,
47   fixval4 INTEGER(0..3),
48   acceptack T-Command-AcceptAck,
49   sourceid  T-Command-SourceID }
50
51 T-Command-ExecAck ::= ENUMERATED {
52   execno (0),
53   execyes (1) }
54
55 T-Command-AcceptAck ::= ENUMERATED {
56   accno (0),
57   accyes (1) }
58
59 T-Command-SourceID ::= ENUMERATED {
60   srcground (0),
61   srcobc (12),
62   srcdpu (20) }
63
64 -- Data-DataFieldHeader
65 T-Data-DataFieldHeader ::= SEQUENCE {
66   fixval3 T-uint8-t,
```

```
67    destinationid T-Data-DestinationID,
68    timestamp T-timestamp }
69
70  T-Data-DestinationID ::= ENUMERATED {
71    destground (0),
72    destobc (12),
73    destdpu (20) }
74
75  -- Command-ApplicationData
76  T-Command-ApplicationData ::= CHOICE {
77    cmd-hk-disable  T-CMD-HK-DISABLE,
78    cmd-hk-enable T-CMD-HK-ENABLE,
79    cmd-hk-period T-CMD-HK-PERIOD,
80    cmd-temp-control-enable T-CMD-TEMP-CONTROL-ENABLE,
81    cmd-temp-control-disable  T-CMD-TEMP-CONTROL-DISABLE }
82
83  T-CMD-HK-PERIOD ::= SEQUENCE {
84    par-sid-hk  T-Command-PAR-SID-HK,
85    par-hk-period T-uint16,
86    pad T-uint8-t }
87
88  T-Command-PAR-SID-HK ::= ENUMERATED {
89    all-hksid (0),
90    default-hksid (1),
91    extended-hksid (2) }
92
93  -- Data-ApplicationData
94  T-Data-ApplicationData ::= CHOICE {
95    dat-event-warning T-DAT-EVENT-WARNING,
96    dat-cmd-execution-failure T-DAT-CMD-EXECUTION-FAILURE,
97    dat-cmd-execution-success T-DAT-CMD-EXECUTION-SUCCESS,
98    dat-cmd-acceptance-failure  T-DAT-CMD-ACCEPTANCE-FAILURE,
99    dat-cmd-acceptance-success  T-DAT-CMD-ACCEPTANCE-SUCCESS,
100   dat-hk-general  T-DAT-HK-GENERAL }
```

```
101
102 T-DAT-HK-GENERAL ::= SEQUENCE {
103   dat-hk-variant  T-DAT-HK-VARIANT }
104
105 T-DAT-HK-VARIANT ::= CHOICE {
106   dat-hk-extended T-DAT-HK-EXTENDED,
107   dat-hk-default  T-DAT-HK-DEFAULT }
108
109 T-DAT-HK-EXTENDED ::= SEQUENCE {
110   hk-temp-fpa-ccd INTEGER(-2147483648..2147483647),
111   hk-temp-fpa-strap INTEGER(-2147483648..2147483647),
112   hk-temp-fee-1 INTEGER(-2147483648..2147483647),
113   hk-temp-fee-2 INTEGER(-2147483648..2147483647),
114   hk-temp-fee-3 INTEGER(-2147483648..2147483647),
115   hk-temp-fee-4 INTEGER(-2147483648..2147483647),
116   hk-volt-fee-1 T-uint16,
117   hk-volt-fee-2 T-uint16,
118   hk-volt-fee-3 T-uint16,
119   hk-volt-fee-4 T-uint16,
120   hk-volt-fee-5 T-uint16,
121   hk-volt-sem-1 T-uint16,
122   hk-volt-sem-2 T-uint16,
123   hk-volt-sem-3 T-uint16,
124   hk-volt-sem-4 T-uint16,
125   hk-volt-sem-5 T-uint16,
126   hk-stat-num-spw-err-credit  T-uint8-t,
127   hk-stat-num-spw-err-escape  T-uint8-t,
128   hk-stat-num-spw-err-disconnect  T-uint8-t,
129   hk-stat-num-spw-err-parity  T-uint8-t,
130   hk-stat-num-spw-err-writesync T-uint8-t,
131   hk-stat-num-spw-err-invalidaddress  T-uint8-t,
132   hk-stat-num-spw-err-eopeep  T-uint8-t,
133   hk-stat-num-spw-err-rxahb T-uint8-t,
134   hk-stat-num-spw-err-txahb T-uint8-t,
```

```
135   hk-stat-num-spw-err-txblocked T-uint8-t,
136   hk-stat-ccd-temp-stable T-Data-HK-STAT-CCD-TEMP-STABLE,
137   hk-stat-fee-temp-stable T-Data-HK-STAT-FEE-TEMP-STABLE}
138
139 T-Data-HK-STAT-CCD-TEMP-STABLE ::= ENUMERATED {
140   ccdtempstable-no (0),
141   ccdtempstable-yes (1) }
142
143 T-Data-HK-STAT-FEE-TEMP-STABLE ::= ENUMERATED {
144   statfeetempstable-no (0),
145   statfeetempstable-yes (1) }
146
147 T-Crc ::= T-uint16
148 END
```

Quellcode 1: Auszug aus der CHEOPS DataView.asn

A.4 Interface View des Beispielprojektes

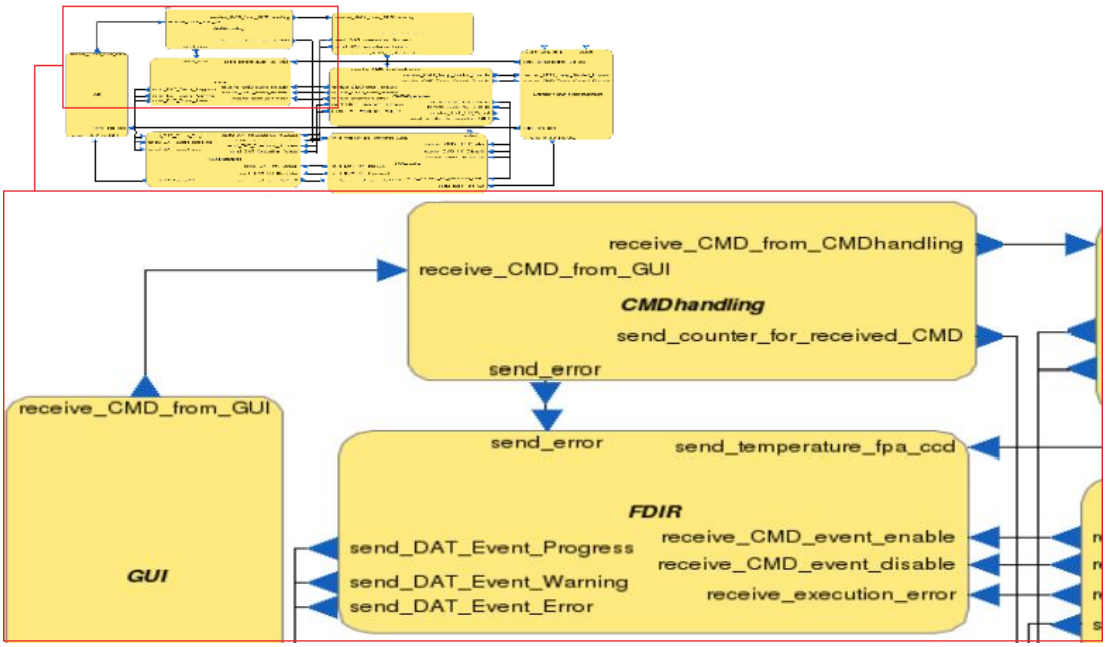


Abbildung 29: Interface View des Beispielsystems (Teil1)

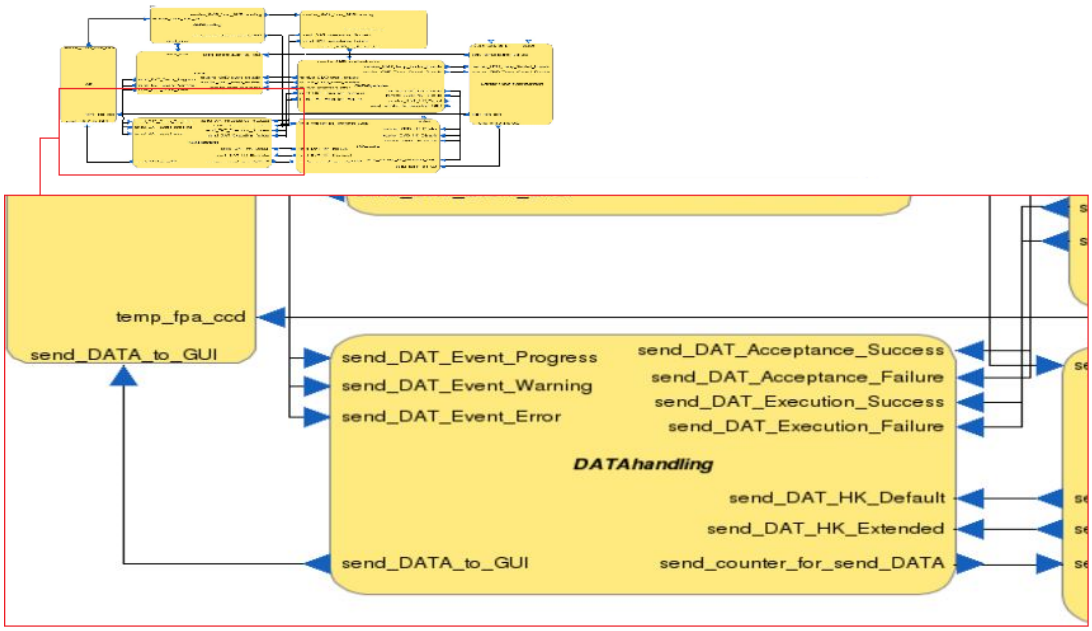


Abbildung 30: Interface View des Beispielsystems (Teil2)

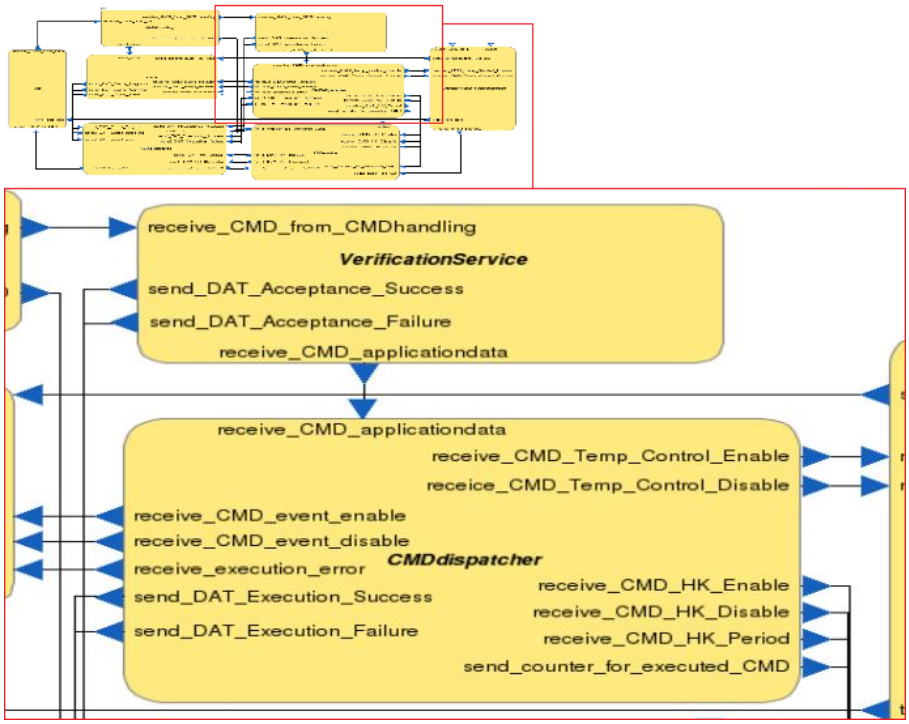


Abbildung 31: Interface View des Beispielsystems (Teil3)

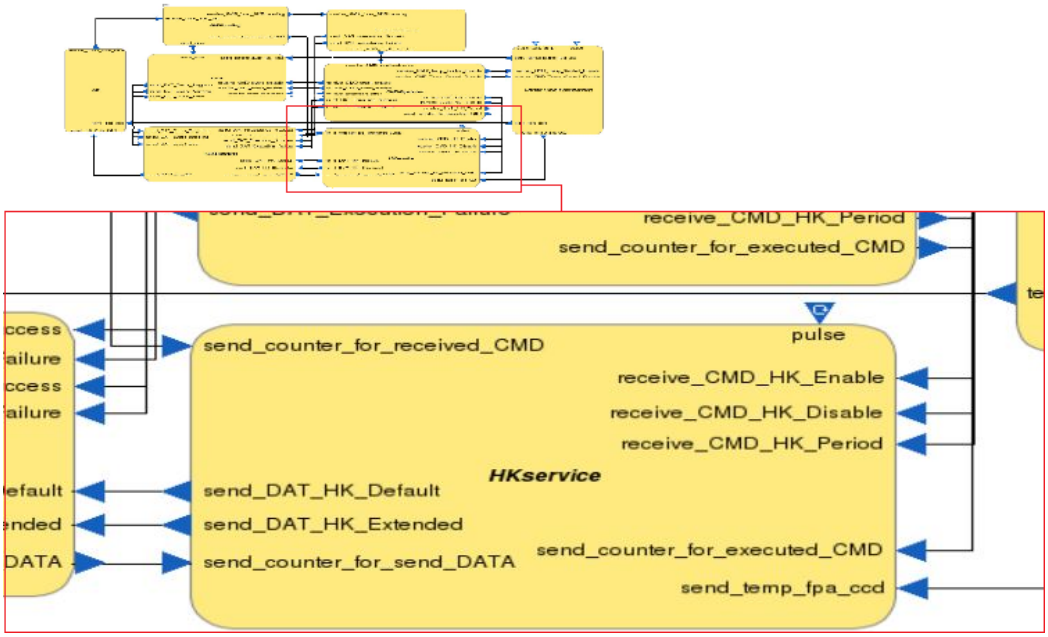


Abbildung 32: Interface View des Beispielsystems (Teil4)

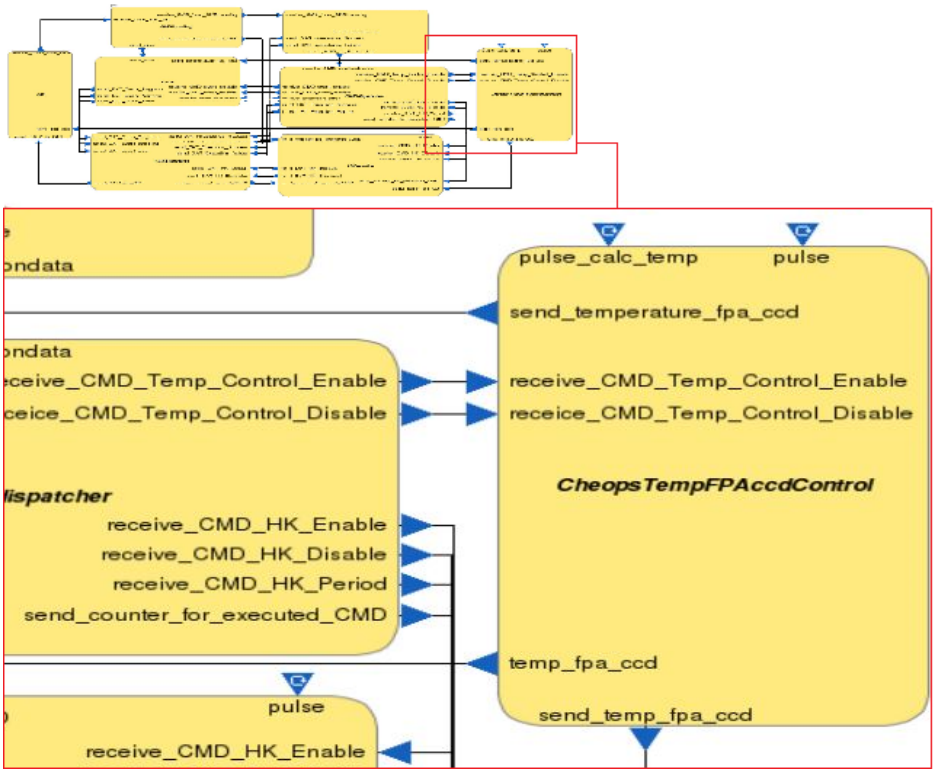


Abbildung 33: Interface View des Beispielsystems (Teil5)

A.5 „VerificationService“-Funktion

```
1 /* Functions to be filled by the user (never overwritten by
      buildsupport tool) */
2 #include <stdio.h>
3 #include <unistd.h>
4 #include "verificationservice.h"
5
6 void verificationservice_startup()
7 {
8     /* Write your initialization code here,
9        but do not make any call to a required interface. */
10 }
11
12 void verificationservice_PI_receive_CMD_from_CMDhandling(const
      asn1SccT_Command_PACKET *IN_cmd_packet)
13 {
14     /* Write your code here! */
15   /*Pack packet*/
16     asn1SccT_Data_PACKET datPacket;
17     asn1SccT_Data_PACKET_Initialize(&datPacket);
18
19     /*PacketHeader*/
20     asn1SccT_Data_PacketHeader datPacketHeader;
21     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
22
23     /*ApplicationData*/
24     asn1SccT_Data_ApplicationData datAppData;
25     asn1SccT_Data_ApplicationData_Initialize( &datAppData);
26
27      /* 1. check APID */
28     if(!(IN_cmd_packet->command_PacketHeader.processid == 60 &&
          IN_cmd_packet->command_PacketHeader.packetcat == 12))
29     {
```

```
30        /*PacketHeader*/
31        datPacketHeader.packetlength = 19;
32        datPacket.data_PacketHeader = datPacketHeader;
33
34        /*ApplicationData*/
35        asn1SccT_DAT_CMD_ACCEPTANCE_FAILURE cmdAcceptanceFailure;
36        asn1SccT_DAT_CMD_ACCEPTANCE_FAILURE_Initialize(&
              cmdAcceptanceFailure);
37        cmdAcceptanceFailure.par_cmd_apid = IN_cmd_packet->
              command_PacketHeader.processid;
38        cmdAcceptanceFailure.par_cmd_sequence_count = IN_cmd_packet->
              command_PacketHeader.sequencecount;
39        cmdAcceptanceFailure.dat_cmd_acceptance_failure_variant.kind =
              dat_cmd_acceptance_failure_wrong_apid_PRESENT;
40        datAppData.kind = dat_cmd_acceptance_failure_PRESENT;
41        datAppData.u.dat_cmd_acceptance_failure = cmdAcceptanceFailure;
42        datPacket.data_ApplicationData = datAppData;
43
44        /* send datcmdacceptancefailurewrongapid to TMhandling*/
45        verificationservice_RI_send_DAT_CMD_Acceptance_Failure(&
              datPacket);
46      }
47       /* 2. check packet length*/
48      /* ASN.1 error: send datcmdacceptancefailureincomplete*/
49      /* 3. check checksum
50       * calculate crc - crc != *IN_tc_packet.commandCrc --> send
            datcmdacceptancefailurewrongcrc */
51      /* 4. check packettype
52       * todo: error: send datcmdacceptancefailurewrongtype */
53      /* 5. check packetsubtype
54       * todo: error: send datcmdacceptancefailurewrongsubtype */
55      /* 6. check sourceID
56       * ASN.1 error: send datcmdacceptancefailurewrongsourceid */
57
```

```
58      else if ( IN_cmd_packet->command_DataFieldHeader.acceptack ==
            asn1Sccaccyes)
59      {
60        /*PacketHeader*/
61        datPacketHeader.packetlength = 15;
62        datPacket.data_PacketHeader = datPacketHeader;
63
64        /*ApplicationData*/
65        datAppData.kind = dat_cmd_acceptance_success_PRESENT;
66        datAppData.u.dat_cmd_acceptance_success.par_cmd_apid =
              IN_cmd_packet->command_PacketHeader.processid;
67        datAppData.u.dat_cmd_acceptance_success.par_cmd_sequence_count =
               IN_cmd_packet->command_PacketHeader.sequencecount;
68        datPacket.data_ApplicationData = datAppData;
69
70        /* send datcmdacceptancesuccess to TMhandling */
71        verificationservice_RI_send_DAT_CMD_Acceptance_Success(&
              datPacket);
72
73        /* send ApplicationData to TCexecution */
74        verificationservice_RI_receive_CMD_applicationdata(IN_cmd_packet
              );
75      }
76      else
77      {
78        /* send ApplicationData to TCexecution */
79        verificationservice_RI_receive_CMD_applicationdata(IN_cmd_packet
              );
80      }
81  }
```

Quellcode 2: VerificationService.c

A.6 „CMDdispatcher“-Funktion

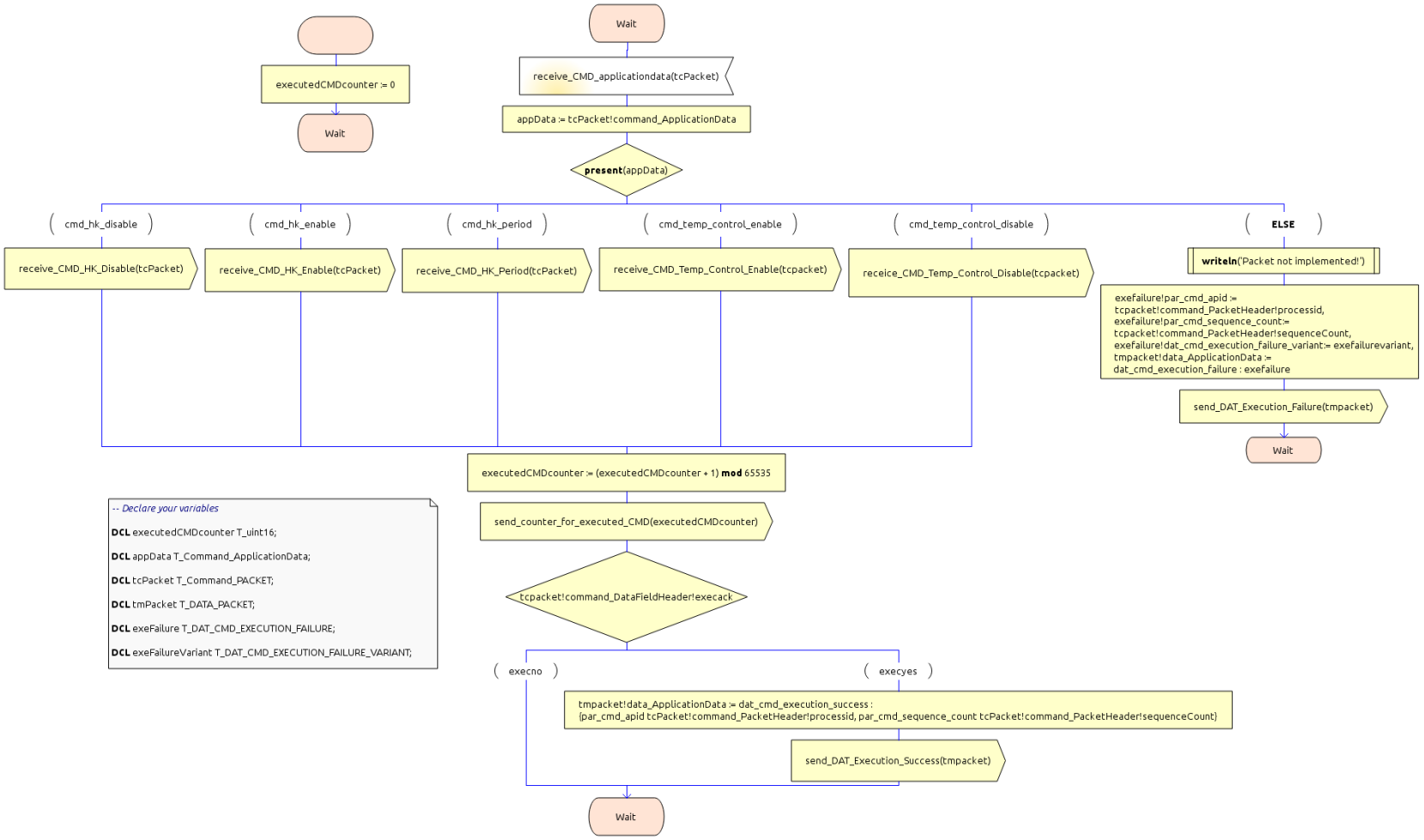


Abbildung 34: SDL-Diagramm der „CMDdispatcher“-Funktion

A.7 „HKservice“-Funktion

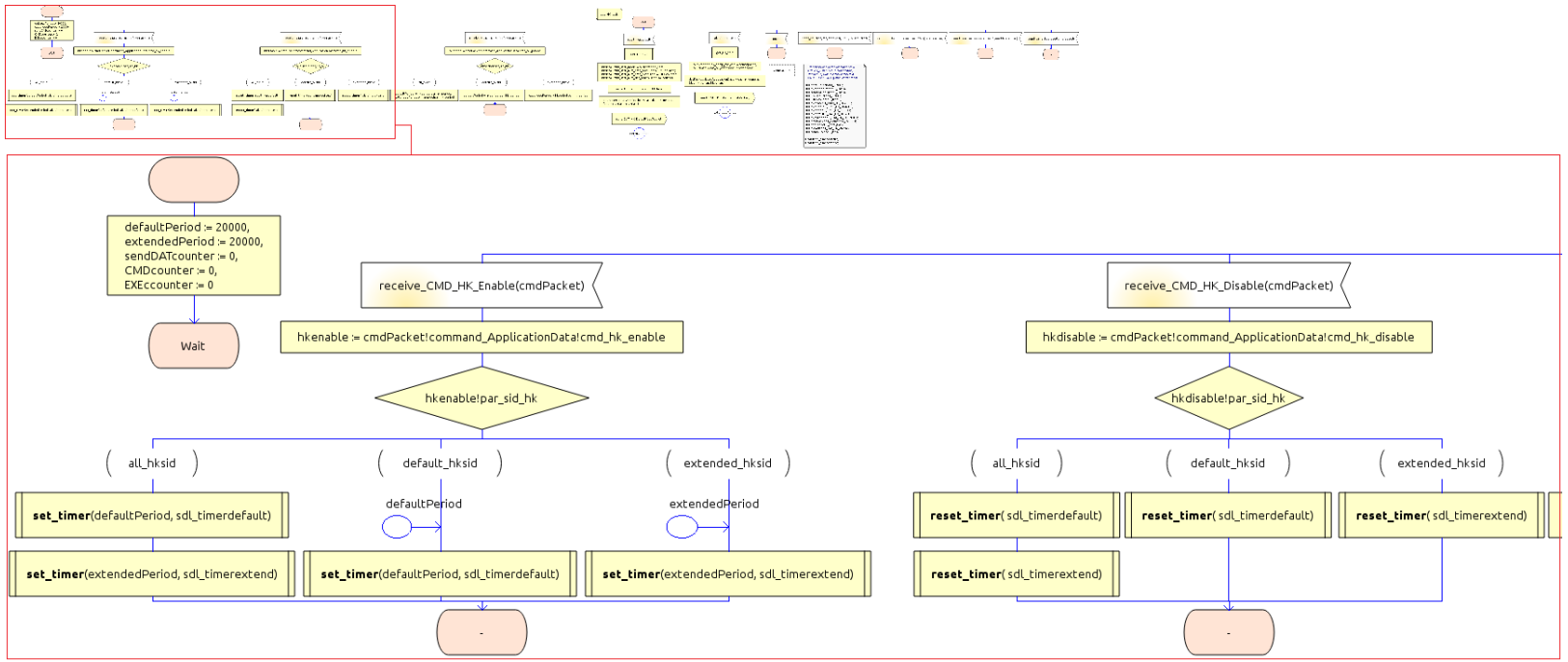


Abbildung 35: SDL-Diagramm der „HKservice“-Funktion (Teil 1)

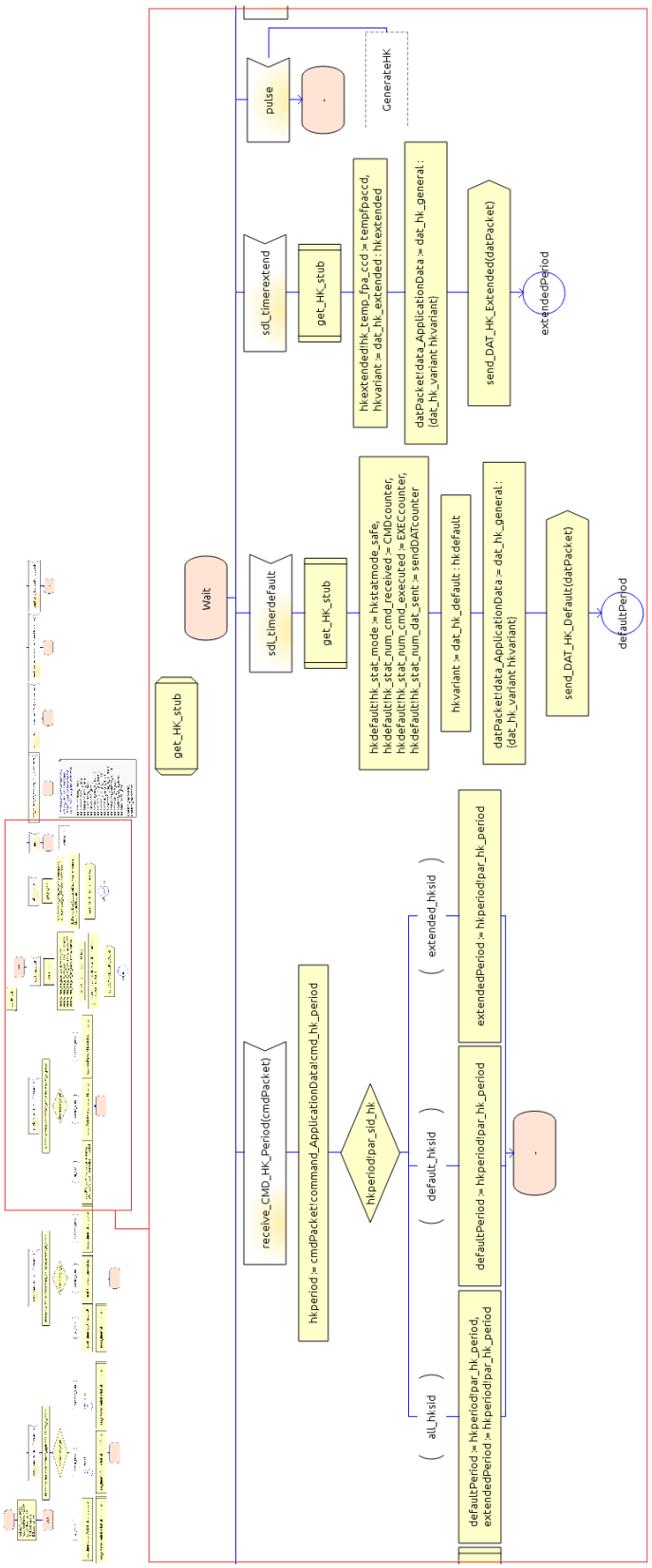


Abbildung 36: SDL-Diagramm der „HKservice“-Funktion (Teil 2)

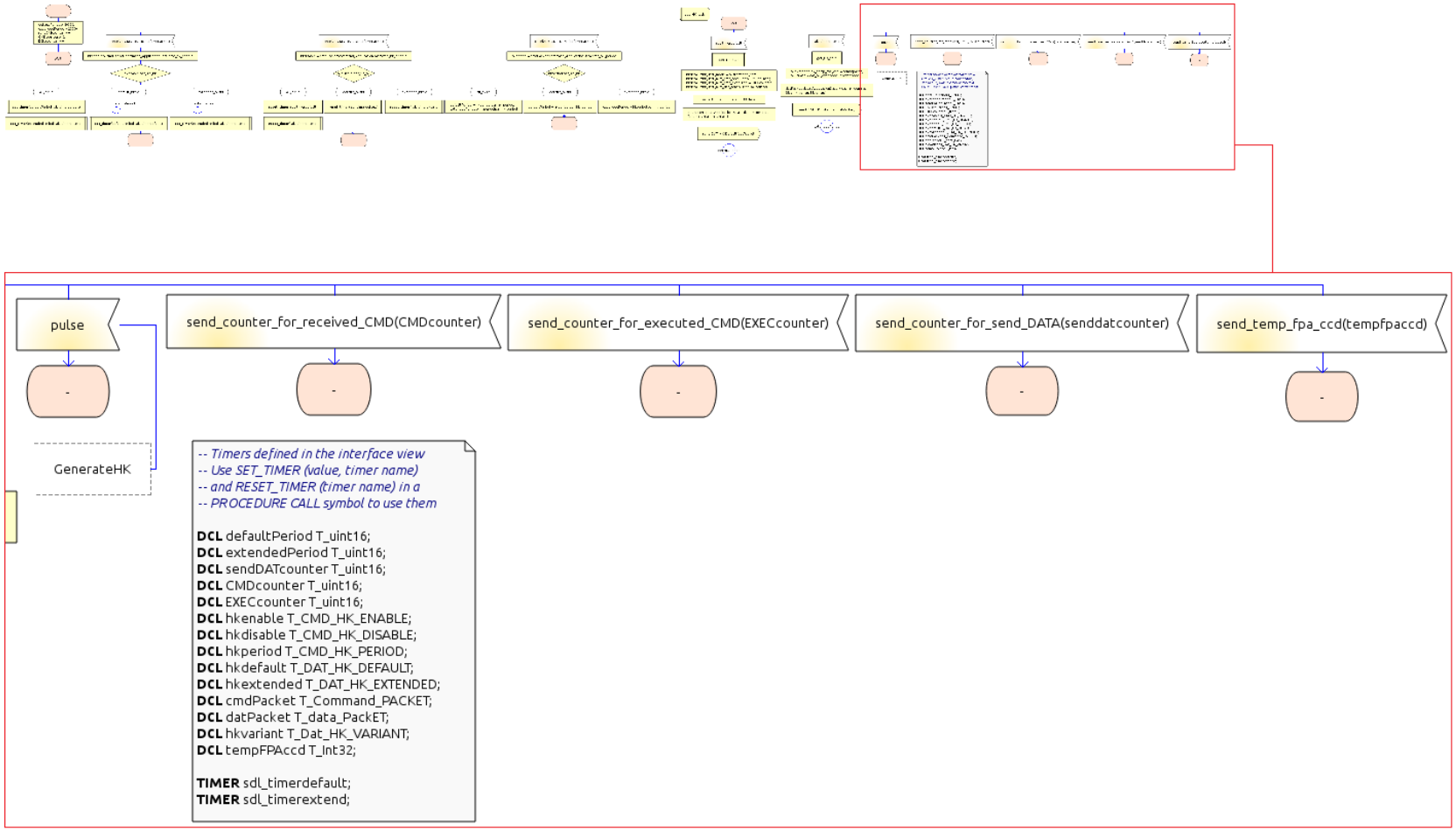


Abbildung 37: SDL-Diagramm der „HKservice“-Funktion (Teil 3)

A.8 „CheopsTempFPAccdControl“-Funktion

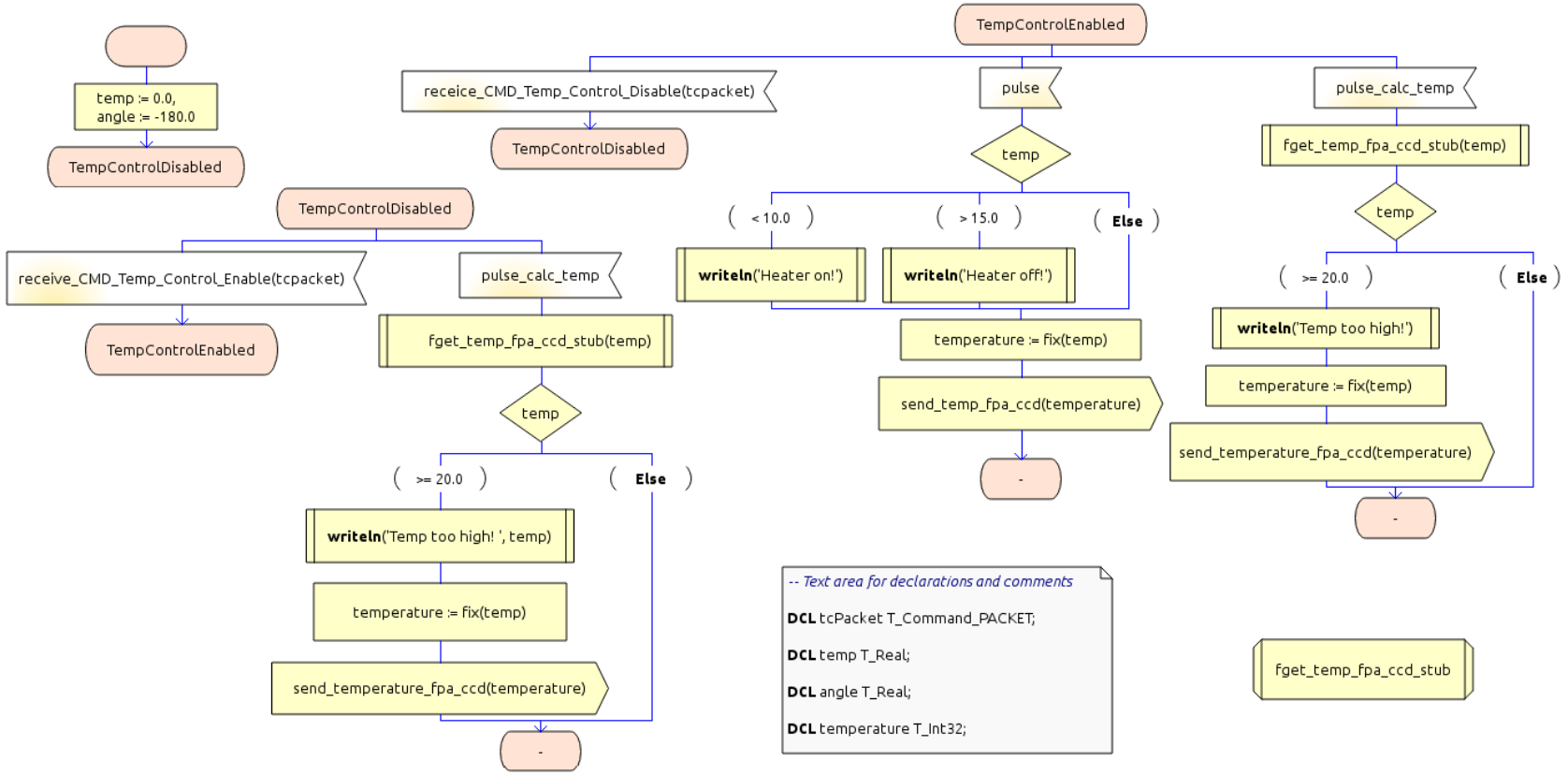


Abbildung 38: SDL-Diagramm der „CheopsTempFPAccdControl“-Funktion

A.9 „FDIR“-Funktion

```
1 /* Functions to be filled by the user (never overwritten by
      buildsupport tool) */
2 #include <stdio.h>
3 #include "fdir.h"
4 void fdir_startup()
5 {
6     /* Write your initialization code here,
7        but do not make any call to a required interface. */
8 }
9
10 void fdir_PI_send_error()
11 {
12     /* Write your code here! */
13 }
14
15 void fdir_PI_receive_CMD_Event_Enable()
16 {
17     /* Write your code here! */
18 }
19
20 void fdir_PI_receive_CMD_Event_Disable()
21 {
22     /* Write your code here! */
23 }
24
25 void fdir_PI_receive_execution_error()
26 {
27     /* Write your code here! */
28 }
29
30 void fdir_PI_send_temp_fpa_ccd_err(const asn1SccT_Int32 *IN_tempfpaccd
      )
```

```
31 {
32     asn1SccT_Data_PACKET datPacket;
33     asn1SccT_Data_PACKET_Initialize(&datPacket);
34 
35     /*PacketHeader*/
36     asn1SccT_Data_PacketHeader datPacketHeader;
37     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
38     datPacketHeader.fixval1 = 1;
39     datPacketHeader.processid = 60;
40     datPacketHeader.packetcat = 1;
41     datPacketHeader.fixval2 = 3;
42     datPacketHeader.sequencecount = 1;
43     datPacketHeader.packetlength = 17;
44     datPacket.data_PacketHeader = datPacketHeader;
45 
46     /*ApplicationData*/
47     asn1SccT_Data_ApplicationData datAppData;
48     asn1SccT_Data_ApplicationData_Initialize( &datAppData);
49     asn1SccT_DAT_EVENT_WARNING_VARIANT eventWarningVariant;
50     asn1SccT_DAT_EVENT_WARNING_VARIANT_Initialize(&eventWarningVariant
           );
51     eventWarningVariant.kind =
           dat_event_warning_fpatemptoohigh_PRESENT;
52     eventWarningVariant.u.dat_event_warning_fpatemptoohigh.
           hk_temp_fpa_ccd = *IN_tempfpaccd;
53     datAppData.kind = dat_event_warning_PRESENT;
54     datAppData.u.dat_event_warning.dat_event_warning_variant =
           eventWarningVariant;
55     datPacket.data_ApplicationData = datAppData;
56 
57     fdir_RI_send_DAT_Event_Warning(&datPacket);
58 }
```

Quellcode 3: FDIR.c

A.10 „DATAhandling“-Funktion

```
1 /* Functions to be filled by the user (never overwritten by
      buildsupport tool) */
2 #include <stdio.h>
3 #include "datahandling.h"
4 void datahandling_startup()
5 {
6     /* Write your initialization code here,
7        but do not make any call to a required interface. */
8 }
9
10 void sendCounter()
11 {
12  static asn1SccT_uint16 dataPacketCounter = 0;
13  dataPacketCounter = (dataPacketCounter + 1) % (1<<16);
14  datahandling_RI_send_counter_for_send_DATA(&dataPacketCounter);
15 }
16
17 void datahandling_PI_send_DAT_CMD_Acceptance_Success(const
      asn1SccT_Data_PACKET *IN_datcmdacksuc)
18 {
19     /* Write your code here! */
20     /* DEFAULT VALUES!!! */
21     asn1SccT_timestamp time;
22     asn1SccT_timestamp_Initialize(&time);
23
24     /*Pack packet*/
25     asn1SccT_Data_PACKET datPacket;
26     asn1SccT_Data_PACKET_Initialize(&datPacket);
27     datPacket = *IN_datcmdacksuc;
28
29     /*PacketHeader*/
30     asn1SccT_Data_PacketHeader datPacketHeader;
```

```
31      asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
32      datPacketHeader.fixval1 = 1;
33      datPacketHeader.processid = 60;
34      datPacketHeader.packetcat = 1;
35      datPacketHeader.fixval2 = 3;
36      datPacketHeader.sequencecount = 1;
37      datPacket.data_PacketHeader = datPacketHeader;
38
39      /*DataFieldHaeder*/
40      asn1SccT_Data_DataFieldHeader dataFieldHeader;
41      asn1SccT_Data_DataFieldHeader_Initialize(&dataFieldHeader);
42      dataFieldHeader.fixval3 = 16;
43      dataFieldHeader.destinationid = asn1Sccdestground;
44      dataFieldHeader.timestamp = time;
45      datPacket.data_DataFieldHeader = dataFieldHeader;
46
47      /*CRC*/
48      asn1SccT_Crc datCRC;
49      asn1SccT_Crc_Initialize(&datCRC);
50      datCRC = 20317;
51      datPacket.data_Crc = datCRC;
52
53      /*send Datapacket*/
54      datahandling_RI_send_DATA_to_GUI(&datPacket);
55      sendCounter();
56  }
57
58  void datahandling_PI_send_DAT_CMD_Acceptance_Failure(const
        asn1SccT_Data_PACKET *IN_datcmdackfail)
59  {
60      /* Write your code here! */
61      /* DEFAULT VALUES!!! */
62      asn1SccT_timestamp time;
63      asn1SccT_timestamp_Initialize(&time);
```

```
64
65      /*Pack packet*/
66      asn1SccT_Data_PACKET datPacket;
67      asn1SccT_Data_PACKET_Initialize(&datPacket);
68      datPacket = *IN_datcmdackfail;
69
70      /*PacketHeader*/
71      asn1SccT_Data_PacketHeader datPacketHeader;
72      asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
73      datPacketHeader.fixval1 = 1;
74      datPacketHeader.processid = 60;
75      datPacketHeader.packetcat = 1;
76      datPacketHeader.fixval2 = 3;
77      datPacketHeader.sequencecount = 1;
78      datPacket.data_PacketHeader = datPacketHeader;
79
80      /*DataFieldHaeder*/
81      asn1SccT_Data_DataFieldHeader dataFieldHeader;
82      asn1SccT_Data_DataFieldHeader_Initialize(&dataFieldHeader);
83      dataFieldHeader.fixval3 = 16;
84      dataFieldHeader.destinationid = asn1Sccdestground;
85      dataFieldHeader.timestamp = time;
86      datPacket.data_DataFieldHeader = dataFieldHeader;
87
88      /*CRC*/
89      asn1SccT_Crc datCRC;
90      asn1SccT_Crc_Initialize(&datCRC);
91      datCRC = 37356;
92      datPacket.data_Crc = datCRC;
93
94      /*send Datapacket*/
95      datahandling_RI_send_DATA_to_GUI(&datPacket);
96      sendCounter();
97  }
```

```
98
99  void datahandling_PI_send_DAT_CMD_Execution_Success(const
        asn1SccT_Data_PACKET *IN_datcmdexesuc)
100 {
101     /* Write your code here! */
102     /* DEFAULT VALUES!!! */
103     asn1SccT_timestamp time;
104     asn1SccT_timestamp_Initialize(&time);
105
106     /*Pack packet*/
107     asn1SccT_Data_PACKET datPacket;
108     asn1SccT_Data_PACKET_Initialize(&datPacket);
109     datPacket = *IN_datcmdexesuc;
110
111     /*PacketHeader*/
112     asn1SccT_Data_PacketHeader datPacketHeader;
113     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
114     datPacketHeader.fixval1 = 1;
115     datPacketHeader.processid = 60;
116     datPacketHeader.packetcat = 1;
117     datPacketHeader.fixval2 = 3;
118     datPacketHeader.sequencecount = 1;
119     datPacketHeader.packetlength = 15;
120     datPacket.data_PacketHeader = datPacketHeader;
121
122     /*DataFieldHaeder*/
123     asn1SccT_Data_DataFieldHeader datDataFieldHeader;
124     asn1SccT_Data_DataFieldHeader_Initialize(&datDataFieldHeader);
125     datDataFieldHeader.fixval3 = 16;
126     datDataFieldHeader.destinationid = 20;
127     datDataFieldHeader.timestamp;
128     datPacket.data_DataFieldHeader = datDataFieldHeader;
129
130     /*CRC*/
```

```
131     asn1SccT_Crc datCRC;
132     asn1SccT_Crc_Initialize(&datCRC);
133     datCRC = 17507;
134     datPacket.data_Crc = datCRC;
135 
136     /*send Datapacket*/
137     datahandling_RI_send_DATA_to_GUI(&datPacket);
138     sendCounter();
139 }
140 
141 void datahandling_PI_send_DAT_CMD_Execution_Failure(const
      asn1SccT_Data_PACKET *IN_datcmdexefail)
142 {
143     /* Write your code here! */
144     /* DEFAULT VALUES!!! */
145   asn1SccT_timestamp time;
146     asn1SccT_timestamp_Initialize(&time);
147 
148     /*Pack packet*/
149     asn1SccT_Data_PACKET datPacket;
150     asn1SccT_Data_PACKET_Initialize(&datPacket);
151 
152     datPacket = *IN_datcmdexefail;
153 
154     /*PacketHeader*/
155     asn1SccT_Data_PacketHeader datPacketHeader;
156     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
157     datPacketHeader.fixval1 = 1;
158     datPacketHeader.processid = 60;
159     datPacketHeader.packetcat = 1;
160     datPacketHeader.fixval2 = 3;
161     datPacketHeader.sequencecount = 1;
162     datPacketHeader.packetlength = 21;
163 
```

```
164     datPacket.data_PacketHeader = datPacketHeader;
165
166     /*DataFieldHaeder*/
167     asn1SccT_Data_DataFieldHeader datDataFieldHeader;
168     asn1SccT_Data_DataFieldHeader_Initialize(&datDataFieldHeader);
169     datDataFieldHeader.fixval3 = 16;
170     datDataFieldHeader.destinationid = 20;
171     datDataFieldHeader.timestamp;
172     datPacket.data_DataFieldHeader = datDataFieldHeader;
173
174     /*CRC*/
175     asn1SccT_Crc datCRC;
176     asn1SccT_Crc_Initialize(&datCRC);
177     datCRC = 17188;
178     datPacket.data_Crc = datCRC;
179
180     /*send Datapacket*/
181     datahandling_RI_send_DATA_to_GUI(&datPacket);
182     sendCounter();
183 }
184
185 void datahandling_PI_send_DAT_HK_Default(const asn1SccT_Data_PACKET *
        IN_dathkdefault)
186 {
187     /* Write your code here! */
188     /* DEFAULT VALUES!!! */
189   asn1SccT_timestamp time;
190     asn1SccT_timestamp_Initialize(&time);
191
192     /*Pack packet*/
193     asn1SccT_Data_PACKET datPacket;
194     asn1SccT_Data_PACKET_Initialize(&datPacket);
195     datPacket = *IN_dathkdefault;
196
```

```
197     /*PacketHeader*/
198     asn1SccT_Data_PacketHeader datPacketHeader;
199     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
200     datPacketHeader.fixval1 = 1;
201     datPacketHeader.processid = 60;
202     datPacketHeader.packetcat = 1;
203     datPacketHeader.fixval2 = 3;
204     datPacketHeader.sequencecount = 1;
205     datPacketHeader.packetlength = 45;
206     datPacket.data_PacketHeader = datPacketHeader;
207
208     /*DataFieldHaeder*/
209     asn1SccT_Data_DataFieldHeader datDataFieldHeader;
210     asn1SccT_Data_DataFieldHeader_Initialize(&datDataFieldHeader);
211     datDataFieldHeader.fixval3 = 16;
212     datDataFieldHeader.destinationid = 20;
213     datDataFieldHeader.timestamp;
214     datPacket.data_DataFieldHeader = datDataFieldHeader;
215
216     /*CRC*/
217     asn1SccT_Crc datCRC;
218     asn1SccT_Crc_Initialize(&datCRC);
219     datCRC = 33085;
220     datPacket.data_Crc = datCRC;
221
222     /*send Datapacket*/
223     datahandling_RI_send_DATA_to_GUI(&datPacket);
224     sendCounter();
225 }
226
227 void datahandling_PI_send_DAT_HK_Extended(const asn1SccT_Data_PACKET *
        IN_dathkextended)
228 {
229     /* Write your code here! */
```

```
230     /* DEFAULT VALUES!!! */
231   asn1SccT_timestamp time;
232     asn1SccT_timestamp_Initialize(&time);
233
234     /*Pack packet*/
235     asn1SccT_Data_PACKET datPacket;
236     asn1SccT_Data_PACKET_Initialize(&datPacket);
237     datPacket = *IN_dathkextended;
238
239     /*PacketHeader*/
240     asn1SccT_Data_PacketHeader datPacketHeader;
241     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
242     datPacketHeader.fixval1 = 1;
243     datPacketHeader.processid = 60;
244     datPacketHeader.packetcat = 1;
245     datPacketHeader.fixval2 = 3;
246     datPacketHeader.sequencecount = 1;
247     datPacketHeader.packetlength = 69;
248     datPacket.data_PacketHeader = datPacketHeader;
249
250     /*DataFieldHaeder*/
251     asn1SccT_Data_DataFieldHeader datDataFieldHeader;
252     asn1SccT_Data_DataFieldHeader_Initialize(&datDataFieldHeader);
253     datDataFieldHeader.fixval3 = 16;
254     datDataFieldHeader.destinationid = 20;
255     datDataFieldHeader.timestamp;
256     datPacket.data_DataFieldHeader = datDataFieldHeader;
257
258     /*CRC*/
259     asn1SccT_Crc datCRC;
260     asn1SccT_Crc_Initialize(&datCRC);
261     datCRC = 47016;
262     datPacket.data_Crc = datCRC;
263
```

```
264     /*send Datapacket*/
265     datahandling_RI_send_DATA_to_GUI(&datPacket);
266     sendCounter();
267 }
268
269 void datahandling_PI_send_DAT_Event_Progress()
270 {
271     /* Write your code here! */
272 }
273
274 void datahandling_PI_send_DAT_Event_Warning(const asn1SccT_Data_PACKET
        *IN_eventwarning)
275 {
276     asn1SccT_timestamp time;
277     asn1SccT_timestamp_Initialize(&time);
278
279     /*Pack packet*/
280     asn1SccT_Data_PACKET datPacket;
281     asn1SccT_Data_PACKET_Initialize(&datPacket);
282     datPacket = *IN_eventwarning;
283
284     /*PacketHeader*/
285     asn1SccT_Data_PacketHeader datPacketHeader;
286     asn1SccT_Data_PacketHeader_Initialize(&datPacketHeader);
287     datPacketHeader.fixval1 = 1;
288     datPacketHeader.processid = 60;
289     datPacketHeader.packetcat = 1;
290     datPacketHeader.fixval2 = 3;
291     datPacketHeader.sequencecount = 1;
292     datPacket.data_PacketHeader = datPacketHeader;
293
294     /*DataFieldHaeder*/
295     asn1SccT_Data_DataFieldHeader dataFieldHeader;
296     asn1SccT_Data_DataFieldHeader_Initialize(&dataFieldHeader);
```

```
297     dataFieldHeader.fixval3 = 16;
298     dataFieldHeader.destinationid = asn1Sccdestground;
299     dataFieldHeader.timestamp = time;
300
301     /*CRC*/
302     asn1SccT_Crc datCRC;
303     asn1SccT_Crc_Initialize(&datCRC);
304     datCRC = 52302;
305     datPacket.data_Crc = datCRC;
306
307    /*send Datapacket*/
308     datahandling_RI_send_DATA_to_GUI(&datPacket);
309     sendCounter();
310 }
311
312 void datahandling_PI_send_DAT_Event_Error()
313 {
314     /* Write your code here! */
315 }
```

Quellcode 4: DATAhandling.c

Literaturverzeichnis

[1] Das DLR im Überblick. http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10443/637_read-251/#/gallery/8570. Version: 10. Juli 2015

[2] Institut für Optische Sensorsysteme. http://www.dlr.de/os/desktopdefault.aspx/tabid-3452/5341_read-7830. Version: 03. September 2015

[3] Fachabteilung Informationsverarbeitung optischer Systeme. http://www.dlr.de/os/desktopdefault.aspx/tabid-3467/5348_read-7937/. Version: 03. September 2015

[4] PERROTIN, Maxime ; TSIODRAS, Thanassis ; DELANGE, Julien ; HUGUES, Jérôme: TASTE Documentation v1.1. 11. Juni 2013

[5] KOSTYANTYN FILONENKO: Model Driven Architecture: Grundlagen und Prinzipien. 2007

[6] PEPPING, Markus: OMG Model Driven Architecture - Grundlagen der MDA. 15. Dezember 2005

[7] CARNEGIE MELLON SOFTWARE ENGINEERING INSTITUTE: Architecture Analysis and Design Language. http://www.aadl.info/aadl/currentsite/. Version: 21. Mai 2015

[8] ULMER, B.: CHEOPS SES Software Design Document. 2.0. 09. Juni 2015

[9] ECSS: Space engineering - Telemetry and telecommand packet utilization. 30. Januar 2003

[10] BERLIN, R. ; PETER, G. ; ULMER, B.: CHEOPS SES Software Development Plan (SDP). 1.0. 12. September 2014

[11] PERROTIN, Maxime ; CONQUET, Eric ; DELANGE, Julien ; TSIODRAS, Thanassis: TASTE: An open-source tool-chain for embedded system and software development. 09. November 2011

[12] PERROTIN, Maxime: taste training: An open-source tool-chain for embedded software development. 04. Februar 2014

[13] DELANGE, Julien: The ASSERT Set of Tools for Engineering. 08. Januar 2011

[14] DELANGE, Julien ; HUGUES, Jérôme ; DISSAUX, Pierre: Validate implementation correctness using simulation: the TASTE approach. 29. November 2011

[15] UNIVERSIDAD DE CANTABRIA: MAST. 24. Juli 2015

[16] PRAGMADEV: PragmaDev - real time modelling and testing tools. http://www.pragmadev.com/product/index.html. Version: 03. Februar 2015

[17] PRAGMADEV: Model your inspiration - Embedded software modelling tool. 2009

[18] TASTE CONSORTIUM: Technical topic: OpenGEODE, an SDL editor for TASTE - TASTE. http://taste.tuxfamily.org/wiki/index.php?title=Technical_topic:_OpenGEODE,_an_SDL_editor_for_TASTE. Version: 02. März 2015

[19] PERROTIN, Maxime: Rapitime Integration. 05. August 2015

[20] WILLERT: IBM Rational Rhapsody - Willert Software Tools GmbH. http://www.willert.de/produkte/model-driven-sw-engineering/ibm-rational-rhapsody-/

[21] IBM Knowledge Center. http://www-01.ibm.com/support/knowledgecenter/SSB2MU_8.0.0/com.ibm.rhp.overview.doc/topics/rhp_c_po_rr_product_overview.html. Version: 01. Januar 2013

[22] IBM - Rational Rhapsody Developer. http://www-03.ibm.com/software/products/de/ratirhap. Version: 10.09.2015

[23] TELELOGIC: TELELOGIC TAU GENERATION2. 02. März 2004

[24] WITTECK, Ulrike: Implementierung eines Schnittstellen Protokoll-Konverters mittels ANS1SCC Kompilers. 31. März 2015

[25] JAN SOMMER ; ULRIKE WITTECK (Hrsg.): AW_ TASTE-PUS: E-Mail. 29. Juli 2015

[26] THE INTERNATIONAL ENGINEERING CONSORTIUM: Specification and Description Language (SDL). 24. Oktober 2002

[27] PERROTIN, Maxime: opengeode: A tiny free, open-source state-machine editor and code generator, based on the SDL and ASN.1 languages. 03. Oktober 2014

[28] PERROTIN, Maxime: TASTE V2 Quick Reference Card. 18. Oktober 2013

[29] TASTE CONSORTIUM: Technical topic: Use of timers in user code with TASTE - TASTE. http://taste.tuxfamily.org/wiki/index.php?title=Technical_topic:_Use_of_timers_in_user_code_with_TASTE. Version: 21. Oktober 2013

[30] FRANK SINGHOFF: The Cheddar project : a free real time scheduling analyzer. http://beru.univ-brest.fr/~singhoff/cheddar/. Version: 04. September 2014

[31] AGNILLERI, Efrem: Data flow analysis for cache optimization in real-time scheduling. 2015

[32] SHA, Lui ; KLEIN, Mark H. ; GOODENOUGH, John B.: Rate Monotonic Analysis for Real-Time Systems

[33] AUDSLEY, N. C. ; BURNS, A. ; RICHARDSON, M. F. ; WELLINGS, A. J.: HARD REAL-TIME SCHEDULING: THE DEADLINE-MONOTONIC APPROACH. 24. November 2001

[34] MAXIME PERROTIN ; ULRIKE WITTECK (Hrsg.): Re_ AW_ TASTE: E-Mail. 05. August 2015